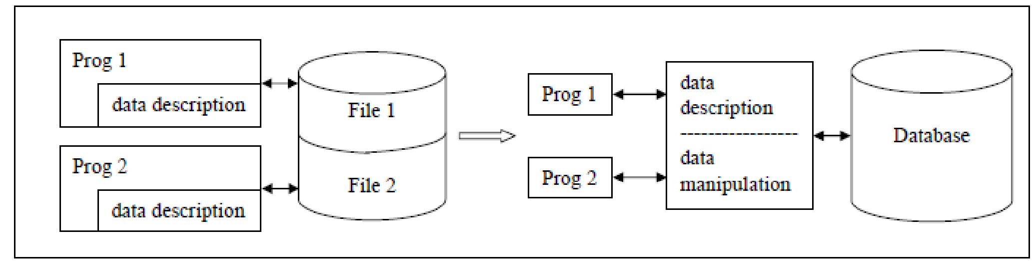
نظم قواعد البيانات الموزعة Distributed Database Systems

نظم قواعد البيانات الموزعة تعتبر خليطا من نظام قاعدة البيانات وتكنولوجيا شبكات الحاسوب، ولأول وهلة يبدو الأمر وكأنه تناقض، حيث أن إدارة قواعد البيانات أساسا حولت التحكم بالبيانات من التطبيقات، كما هو حال الطرق القديمة، إلى التحكم المركزي للوصول إلى البيانات، باستخدام نظام إدارة قاعدة البيانات DBMS ، كما يوضح الشكل التالي.



ومن جهة أخرى تهتم تكنولوجيا شبكات الحاسوب بالتحكم الموزع (غير المركزي non-central)، أما نظم قواعد البيانات فيبدو انها تهتم بالمركزية أساسا، على عكس النظم الشبكية التي تهتم بالتوزيع. ولكننا نعلم أن قاعدة البيانات ليست مهتمة بالفعل بمركزية إدارة البيانات، وذلك لأنها تسمح بدمج البيانات وتقدم طرق مختلفة للوصول إلى البيانات، هذا يعني إمكانية وجود خليط من الشبكات وقواعد البيانات يسمى بنظم قواعد البيانات الموزعة.

وقبل أن نبدأ بالحديث عن هذا الموضوع، دعونا نجيب عن بعض التساؤلات التمهيدية:

ما هي المعالجة الموزعة What is Distributed Processing؟

المعالجة الموزعة تعني أشياء كثيرة لأشخاص/مستخدمين كثيرين، ومن ذلك: الوظائف الموزعة Distributed function (برنامج وحيد يوزع على معالجات متعددة)، أو الحوسبة الموزعة Distributed computing (خدمات مستقلة موزعة على الشبكة)، أو شبكات الحاسوب أو تعدد المعالجات (أكثر من وحدة معالجة مركزية CPU في نفس جهاز الحاسوب)...وغير ذلك من الأمثلة.

لقد تعلمنا عند دراسة مفاهيم نظم التشغيل، أنه في أي جهاز حاسوب يوجد شكل من أشكال المعالجة الموزعة، مثلا وظائف المعالج وأجهزة الإدخال والإخراج.

نحن إذا نحتاج إلى تعريف أفضل للحوسبة الموزعة لفهم حوسبة قواعد البيانات الموزعة.

ما هو نظام الحوسبة الموزع Distributed Computing System؟

نظام الحوسبة الموزعة هو مجموعة من المعالجات المستقلة التي تتواصل فيما بينها بواسطة شبكات الحاسوب، وتتعاون لإنجاز المهام التي تكلف بها، هذه المعالجات يجب أن تكون قادرة على تنفيذ برنامج الحاسوب.

وفي هذا النظام يمكن توزيع أشياء كثيرة، هي:

١. منطق المعالجة Processing logic:

يتم توزيع الآلية التي تتم بها معالجة العمليات المختلفة.

٢. الوظائف Function:

يتم توزيع الخدمات والعمليات الحاسوبية نفسها على شكل إجراءات ودوال محددة.

٣. البيانات Data

يتم توزيع البيانات بتخزينها في مواقع مختلفة.

٤. التحكم Control:

يتم توزيع عمليات السيطرة والتحكم بالبيانات والخدمات الحاسوبية المختلفة.

وفي سياق الحوسبة الموزعة نحن نقصد توزيع كل ما سبق.

درجة الاقتران Degrees of Coupling

يوجد مقياس مستخدم في نظم الحوسبة الموزعة هو درجة الإقتران، ويعرف بأنه نسبة تبادل البيانات إلى كمية المعالجة المحلية المنجزة، ويوجد نوعين رئيسين من الإقتران هما:

الاقتران الضعيف Loose coupling: وهو اقتران يتصف بـ:

- تبادل بيانات قليل نسبيا.
- معالجات تقوم بأغلب عملها بشكل مستقل.

ومن أمثلة ذلك: نظم الاستشعار الحراري.

الاقتران القوي Strong coupling: : وهو اقتران يتصف بـ:

- تبادل بيانات بأحجام كبيرة نسبيا.
- المعالجات تتطلب معلومات كثيرة من بعضها البعض مرات كثيرة.

ومن أمثلة ذلك: الخوارزمية شديدة التوازي.

لماذا نوزع النظم Why Distribute Systems؟

نقوم بالتوزيع من أجل أن تكون النظم أكثر تعويلا reliable، وأكثر استجابة responsive للمستخدم، وهكذا يسير العالم في جميع مناحيه، مثل:

المؤسسات التجارية	Organizational enterprises
التجارة الإلكترونية	E-commerce
مزارع/حقول المخدمات	Server farms
تطبيقات الوسائط المتعددة	Multimedia applications
نظم التحكم الصناعي	Manufacturing control systems

نقوم بالتوزيع أيضا لدعم تكامل البيانات التي قد تكون معقدة بطرق لا تكون مدعومة بنظام وحيد، ولتزويد المؤسسات التجارية بإستقلالية وتحكم ذاتي عبر النظم التي تتجاوب معها، واخيرا نوزع لتقليل تكلفة البرمجيات، فما يخدم موقعا واحدا يمكنه خدمة أكثر من موقع.

إذا، ما هو نظام قواعد البيانات الموزع What is a Distributed Database System

هو مجموعة من قواعد البيانات المتعددة والمترابطة منطقيا، الموزعة عبر شبكات الحاسوب.

أما **نظام إدارة قواعد البيانات الموزعة *Distributed Database Management System (DDBMS)*** *فهو:*

حزمة الأنظمة البرمجية التي تدير قواعد البيانات الموزعة، وتجعل التوزيع شفافا للمستخدم transparent to the user. وهذا المواضيع سوف تغطي الأسس النظرية لتنفيذ الـ DDBMS's.

أشياء ليست من قواعد البيانات الموزعة:

أي مجموعة موزعة من الملفات....a distributed collection of files...لا تعتبر قواعد بيانات موزعة، لأنه يجب ان يكون هناك ترابط منطقي بين بياناتها، فهي تحتاج إلى الكثير من التركيب للوصول إليها ودمجها.

ورغم أن ملفات الـXML ليست ملفات مشوشة ولا غامضة، إلا نسبة الترابط المنطقي فيها غير كافية، حيث تخزن في ملفات متفرقة، كما بنيتها شبه تركيبية semi-structured، بمعنى أنها غير تركيبية تماما مثل قواعد البيانات العلائقية مثلا.

ولهذا فهناك تصور لجعلها اكثر تركيبية وعمل لغة تسمى لغة الاستعلام في الـ XML (xmlql)، وبمجرد تطورها فيمكن أن تكون اكثر تركيبية وقابلة لتحويلها إلى نظم قواعد بيانات موزعة.

أهمية التوزيع الفيزيائي/المادي Physical Distribution

من المهم التركيز على مسألة التوزيع الفيزيائي للبيانات الموزعة منطقيا، حيث أنها قد تكون موزعة في نظم مختلفة different systems، وإذا كانت لدينا قاعدتي بيانات في نفس النظام وكانتا تداران بواسطة نفس نظام إدارة قاعدة البيانات (تجانس)، فيمكن أن تكونا قاعدة بيانات وحيدة single database.

أما لو كان لدينا قاعدتي بيانات في نفس النظام، وكانتا تداران بنظامي إدارة مختلفين (تغاير)، فلا يمكن عندئذ دمجهما في قاعدة بيانات وحيدة، ولكن هذا يضع أمامنا مجموعة من الإشكالات كقضايا الشبكات، وكمون البيانات Latency، وقضايا التغاير heterogeneity (سنأتي لها لاحقا)، وغيرها...الخ.

نظام إدارة قواعد البيانات الموزعة DDBMS ليس نظام متعدد المعالجة multiprocessing system

هناك فرق بين الـ DDBMS ونظام المعالجة المتعددة، فالمسألة مرتبطة بالبيانات نفسها، فحواسيب نظام المعالجة المتعددة تشارك الذاكرة وأقراص بطرق مختلفة، ولكنها تركز على مسالة تنفيذ المعالجات للبرامج، هذه البرامج تحتاج بالطبع للوصول الى البيانات دوريا.

عندما تكون البيانات في الذاكرة المخبأة cache لنظام متعدد المعالجات، تكون لدينا ميزة الوصول السريع Fast access، ولكن هذا سيكون لها تكلفة أكبر Expensive، بسبب ارتفاع ثمن تكنولوجيا

التخزين الرئيسي، مما يؤدي إلى مساحات تخزين محدودة، وضرورة استخدام تقنيات معينة في إدارة الذاكرة والمعالج مثل حذف بيانات بعض وحدات الذاكرة، واستخدام خوارزمية LRU least recently used، وغيرها.

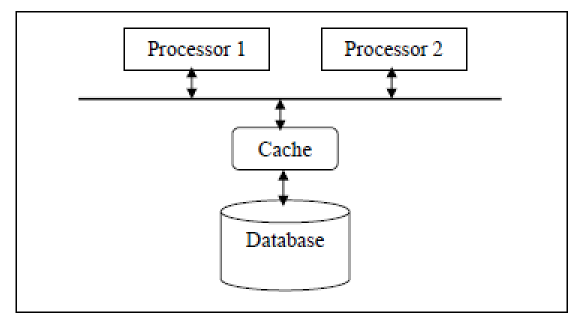
أما عندما تكون البيانات في أقراص التخزين Disks، فالوصول على العكس يكون بطيئا Slow access، ولكن تكلفة هذا النوع من التخزين الثانوي يكون رخيصا Cheap، ولكننا نملك مساحات كبيرة منه.

لدينا إذا في حالة المعالجات المتعددة خيارات ثلاثة:

١. مشاركة الذاكرة Shared Memory (Cache)
٢. مشاركة القرص فقط Shared Disk (Secondary Memory)
٣. عدم مشاركة أي شيء Shared Nothing

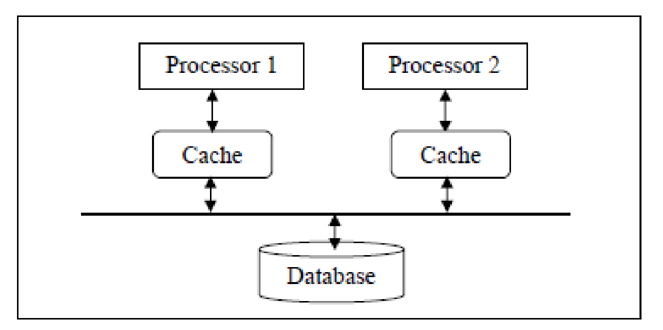
مشاركة الذاكرة Shared Memory (Cache):

كما في الشكل أدناه، تكون لدينا مشاركة في الذاكرتين الأساسية والثانوية، وهذا يعني كفاءة المعالجة عندما تستخدم المعالجات نفس البيانات، ولكن ندخل (عنق الزجاجة Bottleneck)، في حالة كان كل معالج يتطلب بياناته فريدة كل مرة، وهذا يؤدي إلى كثرة التبديل في مواقع الذاكرة much swapping.



مشاركة القرص فقط Shared Disk (Secondary Memory)

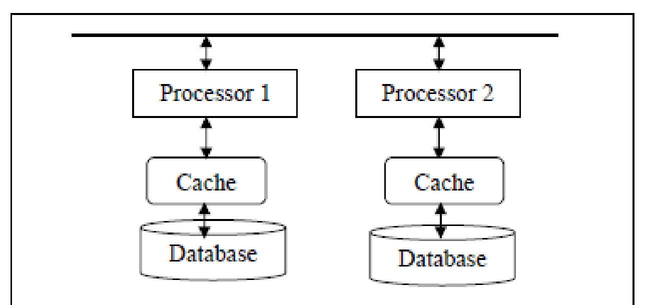
كما في أدناه، لكل معالج ذاكرته الرئيسية الخاصة به، ولكنهما يشتركان بالذاكرة الثانوية فقط، وهذا يؤدي إلى كفاءة في استخدام الذاكرة الرئيسية، ولكن تظهر هنا مشكلة عدم الكفاءة في استخدام المعالجات لنفس البيانات، فعنق الزجاجة يحصل هنا، عندما تتجه المعالجات إلى القرص مرارا.



عدم المشاركة Shared Nothing:

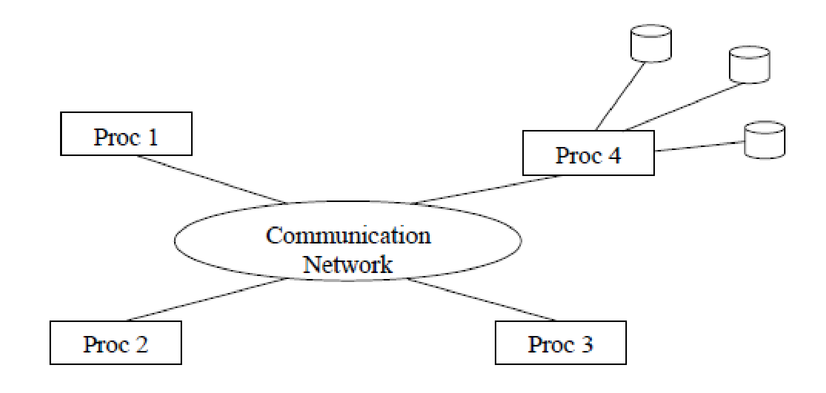
كما في الشكل أدناه، هذه الآلية تكون فعالة عندما تتجه المعالجات لاسترجاع البيانات كل معالج من القرص الخاص بها، وهي فعالة أيضا عندما يذهب المعالج إلى القرص مرات كثيرة.

ولكن عدم الفاعلية يظهر عندما يتجه المعالج إلى استخدام البيانات العامة و/أو البيانات الموزعة common data and/or distributed data.



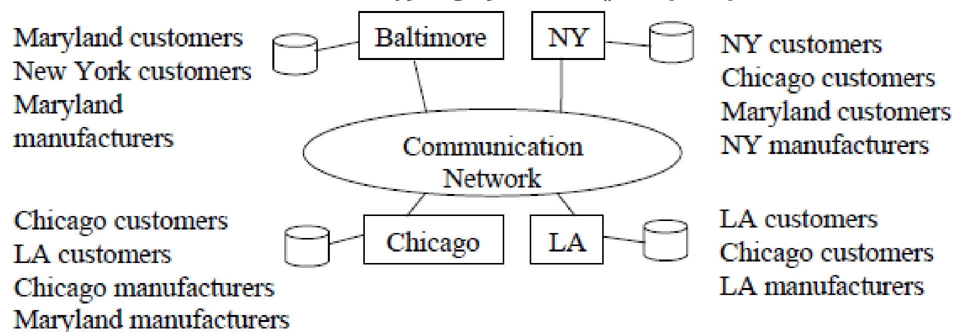
fhdalqasem@yahoo.com فهد آل قاسم

وأخيرا، فنظام قواعد البيانات الموزع ليس قاعدة بيانات مركزية في جهاز واحد، تتصل به مجموعة من النقاط الزبونة كما في الشكل.



إذا، ما هو شكل قواعد البيانات الموزعة حسب التعريف السابق؟

الشكل التالي يوضح مثالا لقواعد بيانات موزعة لمتجر كبير له فروع في الولايات المتحدة، كل فرع يحتوي على بيانات زبائن مدينته وبيانات التصنيع الخاص به، إضافة إلى بيانات فروع مدن أخرى. لدينا هنا قواعد بيانات مستقلة (ذاتيا)، في نقاط/مواقع موزعة.



What a Distributed Database Architecture IS!

قضايا الشفافية Transparency Issues

نعرف الشفافية بأنها عدم تأثير توزيع قواعد البيانات على وظائف المستخدم التقليدية، بحيث يعمل المستخدم على قواعد البيانات الموزعة، دون أن يشعر بمشقة كون قواعد البيانات موزعة، وهذا يجعلنا ندرس الشفافية وفق مجموعة قضايا هي:

شفافية توزيع البيانات Distribution of data وتكرارها Replication of data
شفافية استقلالية البيانات Data independence
شفافية الشبكة Network
شفافية التقسيم Fragmentation

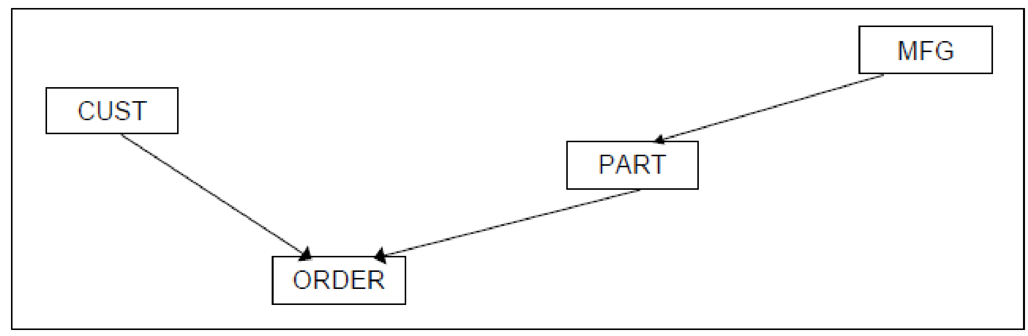
إدارة الشفافية لتوزيع وتكرار البيانات
Transparent Management of Distributed and Replicated Data

على افتراض كان لدينا الجداول التالية لقاعدة بيانات ما:

CUST(cust_no, cust_name, cust_address)
PART(part_no, part_name, manufacturer, cost)
ORDER(cust_no, part_no, quantity)
MFG(manufacturer, state)CUSTORDERPARTMFG

وكانت العلاقة بينها على الشكل التالي:

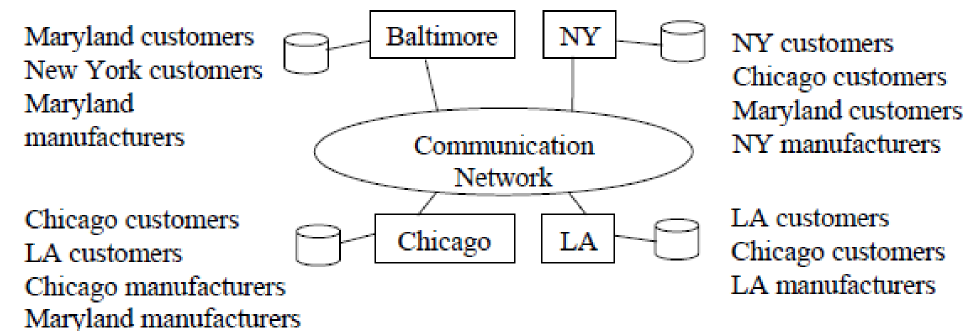
fhdalqasem@yahoo.com فهد آل قاسم



فإن المستخدم حين يستعلم عن بيانات معينة، فهو يكتب كود الـ SQL التالي دون يكون لديه أي شك في صحة هذا الإستعلام، نجاح هذا الإستعلام يعني نجاح إدارة الشفافية رغم توزيع قواعد البيانات، كما أوضحنا سابقا.

```
SELECT cust_name, part_name, quantity FROM CUST, PART, ORDER, MFG
WHERE CUST.cust_no= ORDER.cust_no AND ORDER.part_no = PART.part_no AND
PART.manufacturer = MFG.manufacturer AND
MFG.state = 'MD'
```

دون أن يكون لعمليات المستخدم علاقة بالتوزيع الفعلي التالي لقواعد البيانات:



استقلالية البيانات Data Independence

يتحتم علينا عند توزيع قواعد البيانات أن نمنع تطبيقات المستخدم user applications من أن تغير البيانات بما يتعارض مع استقلالية البيانات المنطقية والفيزيائية.

الاستقلالية المنطقية Logical independence:

وتتمثل بمنع التغيير فالمخطط المنطقي (كإضافة عمود مثلا)، و منع تغيير آلية التوزيع المصممة مسبقا.

الاستقلالية الفيزيائية Physical independence:

أما هذه فتتمثل بمنع التغيير على المخطط الفيزيائي لقاعدة البيانات، وذلك بإخفاء تفاصيل معينة عن بنية تخزين البيانات، وكذلك استقلالية مخطط الفهرسة، وهذا يتحتم استقلالية وسائط التخزين نفسها، وذلك لمنح مرونة في تغيير أي من مخططات الفهرسة ووسائط التخزين دون التأثير على الشفافية، ومن أجل رفع الأداء.

شفافية الشبكة Network Transparency

تحتاج الشبكة إلى أن تحجب عن المستخدمين ما أمكن، وذلك من حيث وجود أمور مهمة كالسرعة وتحميل الشبكة، من الصعب أن يضعها المستخدم في اعتباره عند التعامل مع قواعد البيانات، وكذلك مخطط الشبكة حيث أن له تأثيرا واضحا على إستراتيجية الاستعلام.

أما من وجهة نظر الخدمات Services perspective التي تقدمها الشبكة، فلا حاجة لمعرفة مواقع هذه الخدمات مادامت تؤدي عملها جيدا، كما أن الوصول إلى الخدمات يجب أن يكون موحدا لمن لديهم صلاحية الوصول إليها.

أما من وجهة نظر نظام إدارة البيانات DBMS perspective، فمن المهم عدم ضرورة معرفة موقع البيانات التي يتعامل معها المستخدم، واستقلالية الموقع يعتمد على شفافية التسمية Naming transparency بحيث يطلق المسمى على مواقع مختلفة باعتبار البنية المنطقية لقواعد البيانات لا الفيزيائية.

شفافية التكرار Replication Transparency

يحصل تكرار (وليس ازدواج) مسيطر عليه للبيانات من أجل:

الاستفادة من المرجعية المحلية لبعض البيانات Locality of reference، وهي سمة مستمرة لقواعد البيانات الموزعة، وكذلك من أجل الموثوقية Reliability في البيانات، وايضا من أجل الاحتفاظ بنسخ احتياطية Backup للبيانات.

هل يجب أن يكون التطبيق شفافا؟

يجب بالطبع أن تكون هناك شفافية للتطبيق من جهة المستخدم، أما من جهة النظام ككل، فهناك ضرورة بالنسبة لإدارة المعاملات transaction management أن تكون أبسط عندما يتعامل معها المستخدم، ولكن هذا يؤثر على مرونة الاستخدام لأنه يقلل من استقلالية البيانات، حيث يقوم المستخدم بالتحكم بعدد نسخ البيانات، وكذلك بالتحكم بمواقع البيانات المكررة، وهذا يتعارض مع استقلالية البيانات، كما أسلفنا.

شفافية التقسيم Fragmentation Transparency

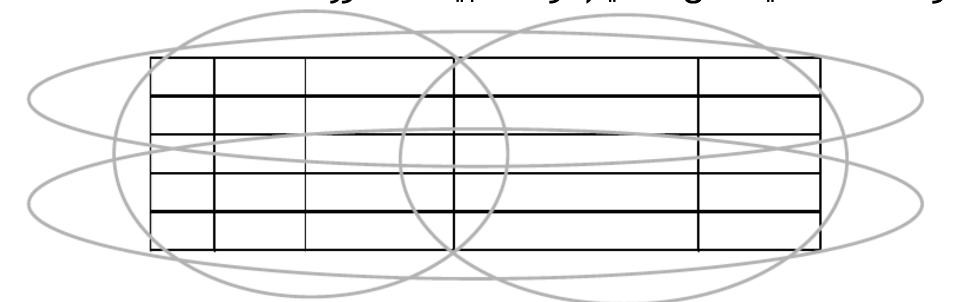
ما يحصل لقواعد البيانات هو توزيع، أما ما يحصل للجداول نفسها في قاعدة البيانات الموزعة، فنسميه بالتقسيم، حيث تقسم وتوزع على قواعد بيانات مختلفة من أجل رفع كفاءة الاستعلام، وهناك ثلاثة أنواع من التقسيم:

التقسيم الأفقي Horizontal

التقسيم العمودي Vertical

التقسيم الهجين بين العمودي والأفقي Hybrid of horizontal and vertical

ما يحدث في الغالب هو التقسيم الهجين، ولكننا سنأخذ كل واحد على حده حتى نفهم التقسيم الهجين جيدا، وذلك عند حديثنا عن تصميم قواعد البيانات الموزعة.



من الذي يجب ان يقدم الشفافية Who Should Provide Transparency؟

من الواضح وجود تعارض بين سهولة الاستخدام وتعقيد التصميم ، فمن ناحية لغة البرمجة تكون هناك مسئولية للشفافية متعلقة بالمترجم المستخدم، حيث أنه لا يمكن فرض شروط الشفافية في المستوى الأدنى.

أما من جهة نظام التشغيل Operating System فمشغلات الأقراص تقدم بعض مستويات الشفافية كخدمات للنظام، وأما من جهة نظام إدارة قاعدة البيانات DBMS فهو يقدم طبقة ربط بين المستخدم/التطبيق ونظام التشغيل، ولكن هذا غير كاف في البيئة الموزعة، ومع هذا فسنكتفي في هذا المقرر بالتركيز على الفوائد التي يقدمها نظام التشغيل لنظام إدارة قواعد البيانات الموزعة DDBMS.

الموثوقية أو المعولية Reliability:

تتلخص المعولية في أربع مواضيع أساسية هي:

المعولية في توزيع المعاملات Distributed transactions

المعولية في تجانس البيانات Consistency

المعولية في تطور الأداء Performance improvements

المعولية في مواجهة التناقضات Complications

المعولية في توزيع المعاملات

Reliability Through Distributed Transactions

تهتم إدارة المعاملات في قاعدة البيانات بضمان أن مجموعة من العمليات البسيطة فيها تنفذ بصورة متتامة أو ترفض كلها، وذلك وفق شروط أو خصائص ACID properties وهي:

Atomicity –all or nothing

الجوهرية ـ إما الجميع وإما لا شيء.

فهد آل قاسم fhdalqasem@yahoo.com

Consistency –effects of a transaction must be repeatable (consistent)
التجانسية ـ يجب أن يصلح تكرار تأثير المعاملة الواحدة على كل الأطرف المتعلقة.
Isolation –transaction operates as if it is the only process interacting with the databases
العزل ـ تعمل المعاملة كأنها وحده التي تتفاعل مع قاعدة البيانات في نفس الوقت.
Durability –effects of transaction last after system crashes
المداومة ـ يجب بقاء تأثير المعاملة حتى بعد خراب النظام واستعادته.
والتكرار يكون في مكونات محددة ومعروفة مسبقا Replicated components، بحيث نضمن أن نقطة وحيدة فاشلة لا تجعل النظام كاملا بصورة غير متاحة للاستخدام، فقط تتأثر عندئذ بعض البيانات، مع ضرورة وصول المستخدم إلى البيانات المتاحة.

المعاملات تضمن التجانسية Transactions Ensure Consistency

يجب ضمان التجانس منذ البدء وحتى نهاية العمليات المختلفة لقاعدة البيانات، وهذا يتضمن العمليات المتزامنة (شفافية التزامن concurrency transparency):
حيث يقوم مجموعة من المستخدمين بتحديث نفس قيم البيانات في نفس الوقت، فيجب أن يضمن لنا مدير المعاملات ان تتصرف التطبيقات وكأن لها وصول حصري إلى نظام قواعد البيانات الموزع.
كما يضمن أن حدوث فشل معين في منتصف العملية، أن يتم تنفيذ كل المعاملة أو عدم أنجاز شيء، ومثال ذلك المحافظة على نقل حسابات الشيكات التجارية.

تجانسية المعاملة Transaction Consistency:

تتطلب تنفيذ ما يسمى ببروتوكولات التحكم بالتزامن الموزع *distributed concurrency control* و بروتكولات معولية التوزيع protocols *distributed reliability*.
النظم التجارية تقدم دجات متخلفة من الدعم لتجانسية المعاملات، مثلا:
نظام الـ Oracle يدعم المعاملات الموزعة distributed transactions ، في حين يدعم نظام الـ Sybase معاملات موزعة بدائية للمستخدمين، وهكذا.....
الأداء المتحسن Improved Performance:

المرجعية المحلية Locality of reference:

يتم استخدام خدمات الـ I/O والمعالجات المحلية من أجل المرجعية المحلية، وذلك من أجل تقليل التأخير الحاصل لمرور الوصول البعيد عبر الشبكة، وبخصوص تقليل ضياع الوقت للبحث عن المحتوى وتقليل الاتصال نستخدم التقسيم للجداول، وتساعد المرجعية المحلية على تحقيق قواعد البيانات الموزعة لدرجة عظمى من المنفعة من التوزيع.

أهمية التوازي Inherent parallelism:

مجموعة متعددة من الاستعلامات تنفذ بالتوازي في أجهزة مختلفة، كما أن أجزاء من نفس الاستعلام يمكن أن ينفذ هو الآخر بشكل متوازي في أجهزة مختلفة.

مواضيع الدراسة في قواعد البيانات الموزعة

تصميم قواعد البيانات الموزعة Distributed Database Design
معالجة الاستعلامات الموزعة Distributed Query Processing
إدارة الدليل الموزع Distributed Directory Management
التحكم بالتزامن الموزع Distributed Concurrency Control
إدارة الورطة الموزعة Distributed Deadlock Management
المعولية في نظم إدارة قواعد البيانات الموزعة Reliability of Distributed DBMS
دعم نظام التشغيل Operating System Support
قواعد البيانات غير المتجانسة Heterogeneous Databases

من الضروري أن نبدأ بالحديث عن معمارية نظم قواعد البيانات الموزعة DDBMS architectures، قبل الدخول في تفاصيل التصميم، وقبل كل هذا سنبدأ بتذكر سريع لمعمارية الحاسوب ومعمارية نظام إدارة قاعدة البيانات.

معمارية الحاسوب Computer Architectures:

- هي التصميم الأولي و البنية الأساسية لعمليات نظام الحاسوب
- وهي توصيف نظري لوظائف الحاسوب مثلا: المتطلبات (وخاصة سرعة الحاسوب والأجهزة المتصلة به)، وتطبيقات التصميم لمختلف أجزاء نظام الحاسوب، مع التركيز ـ بشكل كبير ـ على طريقة انجاز الحاسوب لأعماله.

والكلمة معمارية Architecture يقصد به عادة البنية الداخلية للنظام التي تجري بها العمليات المنطقية، أما تصميم المعمارية Architecture Design فهو مفهوم متعلق بـ:

- التناسب الطردي بين التكلفة والأداء.
- المعولية (أوالوثوقية) Reliability.
- مجموعة الميزات المضافة.
- قابلية التوسيع والتطوير للعتاد Expandability.

معمارية نظام إدارة قواعد البيانات DBMS Architectures :

وهي معمارية مرتبطة ببنية نظام الحاسوب، وتعتمد عادة على نموذج مرجعي reference model، والنموذج المرجعي هو نموذج المعمارية المثالي الذي تسير عليه أغلب نظم إدارة قواعد البيانات العلائقية المعروفة. أو هو إطار عمل أولي (مفاهيمي) يقسم النظام إلى أجزاء قابلة للإدارة، ويوضح بالتالي علاقة هذه الأجزاء مع بعضها البعض.

وبالنسبة لنماذج معمارية نظام إدارة قواعد البيانات، فنحن نستخدم ثلاثة أنواع قياسية (architecture standards) للمعمارية هي:

- معمارية المكونات Component architecture
- معمارية الوظائف Functional architecture
- معمارية البيانات Data architecture

معماريات المكونات Component Architectures:

هي معمارية تقوم بتوضيح مكونات النظام والعلاقات المتبادلة بينها، والمكون component الواحد يقدم للنظام قيمة وظيفية محددة. وبالتفاعل بين المكونات نحصل على القيمة الوظيفية للنظام ككل.

وهذه المعمارية تقدم لنا أفضل طريقة لدراسة أجزاء النظام بغرض بناء النظام من جديد. ولكنها ليست الأفضل من أجل فهم أولي وبسيط للنظام، وفي العادة تقوم مجموعة من المكونات بتشكيل وظيفة واحدة من وظائف النظام.

معماريات الوظيفة Functional Architectures:

من أجل المعمارية الوظيفية نبدأ أولا بتصنيف المستخدمين، وبما أن الوظيفة تكون مرتبطة بالاستخدام، فهذه المعمارية توضح وتعرف الوظائف التي ينجزها المستخدمون، على مختلف تصنيفاتهم.

وبالطبع فتصنيفات وفئات المستخدمين تعرف بالتجزئة الهيكلية decomposed hierarchically والمقصود بها، التجزئة حسب كل مستخدم رئيسي والمستخدمين الفرعيين التابعين له وهكذا.

ومن أمثلة هذه المعماريات، النموذج القياسي المسمى ANSI/SPARC architecture، وهو نموذج قياسي لتعريف نظم إدارة قواعد البيانات.

ومن عيوب هذه المعماريات أنها لا تساعدنا على بناء النظام لأنها لا تساعد أيضا على فهم تعقيد النظام، وإنما تعطي صورة مبسطة للوظائف والمستخدمين.

معماريات البيانات Data Architectures:

هنا نقوم بتعريف طريقة تقديم البيانات والطرق المختلفة لرؤيتها. وهي إطار عمل يساعدنا على فهم كيفية تطبيق النموذج وتحقيقه في الواقع، ومنذ كان نظام إدارة قواعد البيانات DBMS مركزيا، فقد كانت

هذه المعمارية هي الخيار المقدم، ولكن لابد من اجل فهم كامل للنظام أن نقوم بعرض المعماريتين السابقتين، الوظيفة والمكونات، وذلك من أجل رؤية كاملة للنموذج.
أنظر الشكل في الصفحة ٧ الذي يوضح:
المعمارية المرجعية لنظام إدارة قواعد البيانات DBMS حسب لجنة ANSI/X3/SPARC.
ANSI/X3/SPARC Committee DBMS Reference Architecture

أمثلة توضيحية لـ ANSI/X3/SPARC Committee DBMS Reference Architecture:

- المنظور الخارجي External views هي مشاهد للبيانات المشتركة بين المستخدمين والتطبيق الذي يتعامل مع نظام إدارة البيانات، والتي نستخدم جملة SQL التالية لتنفيذ مثال لها:

```
CREATE VIEW GOOD_CUST AS
 SELECT DISTINCT cust_name, cust_address FROM CUST, ORDER
WHERE CUST.cust_no= ORDER.cust_no AND ORDER.quantity > 10
```

- و المنظور الأولي (المفهومي) فهو منظور العالم الكلي لبيئة البيانات، ويعبر عنه بمخطط الإسكيما DB schema حسب المثال التالي:

```
CUST (cust_no, cust_name, cust_address) key is cust_no
```

- أما المنظور الداخلي فهو متعلق برؤية قاعدة البيانات لبنية وتركيب البيانات فيها، كما يوضحه المثال التالي:

```
CUST( index on C#;
C# : 4 bytes,
C-name: 30 bytes,
C-addr: 100 bytes )
```

حيث ان العمود C# هو المفتاح الرئيسي ويقبل ٤ بايتات من البيانات، والعمود C-name فيقبل ٣٠ بايت وهكذا..

رسم تخطيطي جزئي لمعمارية ANSI/SPARC:

يوضح الشكل في الصفحة ١٠ رسما تخطيطيا جزئيا لمعمارية ANSI/SPARC، بحيث أن كل المناظير المقدمة المختلفة عنه يتم توضيحها في قاموس البيانات، والذي قد لا يعرض في هذا المخطط.
في المخطط أن قاعدة البيانات الأولية Conceptual DB تكون من مهام مدير المؤسسة / مدير النظام Enterprise administrator والذي يقوم في الواقع بتصميم قاعدة البيانات الأولية، أما مدير قاعدة البيانات DB administrator والذي يرمز له اختصارا بـ (DBA)، فيقوم بـ:

- الاهتمام بقاعدة البيانات الداخلية Internal DB.
- يتأثر ويستخدم قاعدة البيانات الأولية Conceptual DB.

أما مدير التطبيقات Application administrator فيقوم بـ:

- الاعتناء بقاعدة البيانات الخارجية External DB.
- يتأثر ويستخدم قاعدة البيانات الأولية.

وبالنسبة لقاموس أو دليل البيانات فمهامه تتلخص بـ:

- الاهتمام بسلامة المخطط ككل، وذلك بعدم تغيره إلا حسب الأنسب للنظام.
- تحويل وترجمة مختلف أساليب العرض والتواصل مع المستخدمين.
- تعريف بنية وتركيب قواعد البيانات الداخلية والخارجية للتطبيقات المختلفة.

الرسم التخطيطي لـ Partial Schematic of ANSI/SPARC:
أنظر صفحة ١٠.

نماذج معمارية قواعد البيانات الموزعة Architecture Models for Distributed DBs:
نقوم بتوصيف معمارية قواعد البيانات الموزعة وفق ثلاثة تصنيفات تقسم على أساسها وتوصف هذه المعمارية، وذلك حسب:
الاستقلالية Autonomy: أي نوع ومدى استقلالية البيانات.
التوزيع Distribution: أي مستوى التوزيع المطلوب Level of distribution .
التجانسية Heterogeneity:قواعد البيانات الموزعة متجانسة؟ (heterogeneous\homogenous).

تعريفات الاستقلالية Definitions of Autonomy:
الكلمة Autonomy استقلالية تختلف عن الاستقلال independence التي تعتبر الدرجة القصوى من درجاتها، ولهذا فلها تعريف مختلفة، موضحة صفحة ١٢، ولكننا سنكتفي بالتعريف الرئيسي، تعريف الدكتور تامر أوتسو.
تعريف الاستقلالية Ozsu's Definition of Autonomy:
تصنف استقلالية قواعد البيانات الموزعة إلى ثلاثة حالات:
الحالة صفر: التكامل المحكم : صورة وحيدة تكون متاحة لقاعدة البيانات، بدمج جميع القواعد الموزعة فيها.
0. Tight integration--single image of DB available
الحالة واحد: شبه أو نصف مستقلة وفيها تكون قواعد البيانات قد حددت الجزء منها الذي ترغب بمشاركته، وهو الجزء الذي يجب تعديله من أجل تبادل المعلومات بين القواعد الأخرى.
1. Semi-autonomous
•Databases determine what parts of database they want to share
•Must be modified to exchange information with each other
الحالة اثنين: العزل الكلي وفيها تكون قواعد البيانات معزولة عن بعضها البعض بدون دمج.
2. Total isolation-stand-alone databases
توزيع قاعدة البيانات Database Distribution:
يكون التوزيع كما الاستقلال على ثلاثة حالات:
الحالة صفر حيث لا يوجد توزيع
0. None
الحالة واحد وهي حالة توزيع للوظائف فتكون بعض المواقع مخدمة والأخرى زبونة، حالة المخدم/الزبون.
1. Client/Server --distributes functionality of DB
الحالة اثنين وهي حالة تكون الوظيفة موحدة بين المواقع حالة الند للند، وفيها يكون التوزيع تاما، والتفاعل متناسق بين الجميع.
2. Peer-to-peer
–Fully distributed
–Act in concert with each other
التجانسية/عدم التجانسية Heterogeneity:
وهي حالتان فقط:
إما تجانس وهي الحالة صفر. والتجانس هو وجود نفس مدير نظام قواعد البيانات الموزع لجميع المواقع.
0. Homogeneous
أو عدم تجانس وهي الحالة واحد.
1. Heterogeneous
والتي يقصد بها مجموعة من الحالات التالية أو إحداها أو بعضها.
أجهزة الكترونية مختلفة Different hardware
نماذج بيانات مختلفة Different data models
لغات استعلام مختلفة Different query languages
نماذج المعاملات/العمليات المختلفة Different transaction models

Examples:

•A0,D2,H0	
–Tightly integrated system	نظام متكامل بصورة محكمة – عدم الاستقلالية
–DBMSs located peer-to-peer	نظام إدارة قواعد بيانات محلي الند للند
–Same access, platforms, etc.	تجانس في الوصول ومنصة والعمل ..الخ
•A1,D0,H1	
–Semiautonomous --different type of data (video & text)	قواعد البيانات الموزعة نصف مستقلة
–No distribution	لا يوجد توزيع لـ DBMS
–Heterogeneous access	وصول غير متجانس
–Heterogeneous, federated DBMS	نظام DBMS متحد غير متجانس
•A2, D1, H1	
–Autonomous database systems	نظم قواعد البيانات مستقلة
–Client/server architecture	معمارية المخدم/الزبون
–Heterogeneous access	وصول متجانس
–Functionality in middleware –three-layer architecture	وظائف متوسطة ضمن معمارية ثلاثة الطبقات

معماريات شهيرة لقواعد البيانات الموزعة:

فيما تبقى نتحدث عن ثلاثة معماريات شهيرة لقواعد البيانات الموزعة وهي:

Client/Server Architecture (A*, D1, H*)	معمارية المخدم/زبون
Peer-to-Peer Architecture (A0, D2, H*)	معمارية الند - لـ - الند
Multiple Database Architecture (A2, D*, H*)	معمارية قواعد البيانات المتعددة

معمارية المخدم/زبون Client/Server Architecture (A*, D1, H*):

وهي معمارية تأخذ أي واحدة من حالات الاستقلالية الثلاث، وأي واحدة من حالتي التجانسية، ولكنها تكون في حالة التوزيع الثانية، حالة المخدم/زبون، والتي فيها يكون هناك مواقع تدير عمليات البيانات، ومواقع تستخدم البيانات فقط بدون إدارة، بغض النظر عن موقع قواعد البيانات الموزعة ومدى استقلاليتها أوتجانسها.

ويمكن لمعمارية المخدم/زبون أن تأخذ إحدى حالتين:

١. زبائن عديدة ومخدم وحيد Multiple-client/single-server:

وهي حالة تشبه وجود نظام إدارة قواعد البيانات الموزعة في جهاز واحد، مع اختلافات تراعي فقط مسألة إدارة المعاملات/العمليات.

٢. زبائن عديدة/مخدمات عديدة Multiple-client/multiple-server:

وهذه المعمارية يمكن أن تنفذ في أسلوبين، إما أن تنظم التطبيقات الوصول بين المخدمات والزبائن Application manages data access، وفي هذه الحالة يكون العبء على مواقع الزبائن، أو أن ينظم هذه العملية واحد من المخدمات، بحيث يدير طلبات المعلومات المخزنة في المخدمات الأخرى، وهنا يكون العبء على المخدم المخصص لذلك، ويكون لدينا حالة الزبائن الخفيفة "light clients".

ومهما كان نوع المعمارية فالمستخدم يتعامل معها كأنها شيء واحد، فالفارق بين معماريتي الند للند والمخدم/زبون لا يكمن إلا في المعمارية.

صفحة ١٨ شكل يوضح معمارية المخدم/زبون.

معمارية الند للند Peer-to-Peer Architecture (A0, D2, H*):

كما هو واضح من الترميز (A0, D2, H*)، تكون قواعد البيانات هذه المعمارية محكمة التكامل، وتكون موزعة تماما كما في الحالة الثالثة للتوزيع، ويكون لها أي واحدة من حالتي التجانس.

مميزاتها Features:

Data independence	استقلالية البيانات
Network transparency -supported by global schemas & mapping	يتم دعم شفافية الشبكة بمخططات وتحويل على المستوى العام
Users query independent of location	استعلام المستخدمين مستقل عن الموقع
Global mapping taken care of at GCS level	التحويل بالمستوى العام يأخذ بالإعتبار مستوى المخطط الأولي العام
Local mapping taken care of at LCS/LIS level	التحويل المحلي يأخذ بالاعتبار في المستوى LCS/LIS

صفحتي ٢٠، ٢١ شكل يوضح معمارية الند للند.

عناصر معمارية الند للند Peer-to-Peer Architecture Elements:

- معالج واجهة المستخدم UIF: يفسر أوامر المستخدم.
- المتحكم الدلالي بالبيانات SDC: يفحص سلامة الشروط/القيود، والصلاحيات، والصياغة..الخ.
- محسن ومحلل الإستعلامات العام: يقلل من تكلفة الاستعلامات العامة، ويجد أفضل الخطط، ..الخ.
- مراقب التنفيذ الموزع: ينسق التنفيذ الموزع للطلبات، ويدير المعاملات الموزعة، ..الخ.
- معالج الاستعلامات المحلية: يختار افضل مسارات الوصول عبر الشبكة، ويقلل من تكلفة الاستعلام، ويجد افضل الخطط، الخ.
- مدير الإستعادة المحلي: يحافظ على سلامة وتجانس قاعدة البيانات.
- دعم وقت-التنفيذ: هي برامج وإجراءات نظام التشغيل التي تتفاعل مع ملفات بيانات قاعدة البيانات.

معمارية قاعدة البيانات المتعددة Multiple Database Architecture (A2, D*, H*)

مهما كانت طبيعة التوزيع والتجانس، فقاعدة البيانات المتعددة هي قواعد بيانات موزعة بالحالة الثالثة، حالة الند للند، ورغم انها واحدة من معماريات القواعد الموزعة إلا أن MDBMS يختلف عنها عموما بالتالي:

نظام إدارة قواعد البيانات المتعددة MDBMS:

تصميمه يكون بأسلوب من أسفل لأعلى، ويقوم الـ GCS بوصف بعض قواعد البيانات، كما أنه يعتبر جزء من قواعد البيانات.

نظام إدارة قواعد البيانات الموزعة Distributed DBMSs:

يكون تصميمه عادة بأسلوب من أعلى لأسفل، ويقوم الـ GCS، بوصف جميع قواعد البيانات، كما أنه يعتبر اتحادا لقواعد البيانات الموزعة هذه.

خلاصة مهمة Conclusion:

معمارية البيانات تقدم إطار عمل لنظم قواعد البيانات الموزعة، وتركز هذه المعماريات على:

User views	منظور المستخدم
Data views	منظور البيانات
Levels of autonomy	مستوى الإستقلالية
Levels of distribution	مستوى التوزيع
Levels of heterogeneity	مستوى التجانسية

وعدم فهم هذه القضايا سوف يؤثر سلبا على أمرين هما: التصميم وخوارزميات معالجة الاستعلامات الموزعة.

تصميم التوزيع Distributed Design:

لو كنا بصدد تصميم البرامج الموزعة لكان اهتمامنا منصبا حول أين نضع البرامج والبيانات الموزعة، وهذا يعني ان نتحدث عن تصميم موقع البرامج التطبيقية وموقع نظم إدارة قواعد البيانات، ولكن من الواضح أن هذا ليس مجالنا.

في تصميم قواعد البيانات الموزعة، سنهتم بتنظيم البيانات، والذي يقصد به، كيفية تجزئة البيانات، وأين نضع أجزاء البيانات هذه؟. كل هذا بهدف جعل الوصول إلى البيانات أسرع و أكثر كفاءة، وذلك من خلال عمل مرجعية محلية *locality of reference* للبيانات الموزعة، ووضع البيانات التي يحتاجها المستخدم بشكل دائم، في مواقع قريبة منه.

مستويات المشاركة Levels of sharing:

يقصد بالمشاركة، التعامل مع نسخة واحدة، دون تكرارها (عمل نسخ منها في مواقع أخرى)، وبالنسبة لمستويات مشاركة البيانات والبرامج، فنحن إما أن لا نشارك أي من البيانات والبرامج، أو أن نقوم بمشاركة البيانات فقط، أو أن نشارك الجميع، البيانات والبرامج معا. والحالة الأولى No sharing ليست صالحة للتطبيق في بيئات البيانات المعقدة والمركبة.

أما الحالة الثانية Data sharing only، فنشارك البيانات فقط، ولكننا قد نكرر البرامج عند الضرورة. أما في حالة Program and data sharing مشاركة الجميع، فلا يتم تكرار أي من البيانات أو البرامج.

سيتم التركيز على المعماريات التي تدعم الحالتين الأخيرتين من المشاركة، أي حالة مشاركة البيانات، وحالة مشاركة الجميع.

أنماط الوصول Access Patterns:

من المهم فهم أنماط الوصول للمستخدمين والتطبيقات، وذلك من اجل فهم أي واحد من المستخدمين محتاج لأي من البيانات الموزعة. ولمعرفة ما هي أنواع البيانات التي تناسب أنواع احتياجات المستخدمين المختلفة؟، وكذلك أين يقع المستخدمين بالنسبة للبيانات المطلوبة.

أنماط الوصول الساكنة Static access patterns:

هي أنماط تتصف بأنها ليست مطلوبة دائما، وهنا نبدأ مباشرة في تصميم وإدارة بيئة البيانات الموزعة.

أنماط الوصول الديناميكية Dynamic access patterns:

من المحتمل جدا، أن كثير من المستخدمين لا يملكون نفس الاحتياج من البيانات طوال الوقت. ويكون من الصعوبة الكبيرة توقع احتياجاتهم، وفي هذا النمط يكون من الصعب تصميم وإدارة بيئة البيانات الموزعة. لهذا سنهتم في البدء بدراسة أنماط الوصول الساكنة فقط. فالطرق الساكنة ستدرس من اجل خدمة الطرق الدينامكية الأكثر تعقيدا.

ما نعرفه عن أنماط الوصول Knowledge About Access Patterns:

ما مستوى معرفتنا لإجابة السؤال : كيف سيصل المستخدمون للبيانات؟، إن ما نعرفه عن ذلك يمكن أن يكون في المدى التالي:

- لا معرفة (من الصعوبة معرفة كيفية توزيع البيانات).
- معرفة جزئية.
- معرفة تامة تساعدنا على تحديد مثال لمواقع البيانات المفترضة.

إن المعرفة الجزئية، في الغالب، هي الحالة الأكثر احتمالا، بحيث نقوم بأفضل ما يمكننا فعله مبدئيا، فنحن سنقوم بمراقبة أنماط الاستخدام، مع مرور الوقت، للوصول إلى أفضل تصور عن أنماط الوصول للبيانات.

كل ما سبق يساهم في تصميم وتحديد مواقع البيانات الموزعة.

استراتيجيات التصميم Design Strategies:

١. إستراتيجية من أعلى لأسفل Top Down:

تستخدم هذه الإستراتيجية في حالة كان مصممو قواعد البيانات لديهم تحكم كلي (سلطة مطلقة) في تصميم كامل قاعدة البيانات، وفي توزيع البيانات، وهذا لا يحدث إلا للمصمم المحظوظ في حالات خاصة.

٢. إستراتيجية من القاع للقمة Bottom Up:

على عكس الإستراتيجية السابقة، نستخدم هذه الإستراتيجية عندما يكون المصممون بدون أي سلطة لتصميم وتوزيع قواعد البيانات. وهذا يحدث للمصمم سيئ الحظ في حالات خاصة أيضا.

٣. الإستراتيجية المختلطة (الهجين) Hybrid:

يكون للمصممين هنا بعض السلطات على التصميم وأنماط التوزيع، وهذه هي الحالة الملاحظة في أغلب الوقت، وفهمها يتطلب فهم الإستراتيجيتين السابقتين من أجل المزج بينهما حسب الحاجة، في هذه الإستراتيجية.

التصميم من أعلى لأسفل Top Down Design:

التصميم الأولي للبيانات في هذه الحالة هو نموذج ER، نموذج كينونة-علاقة، المصمم للمنظمة كلها. والذي من خصائصه أنه يوحد بين المناظير المختلفة، مع ضرورة توقع استخدامات أو رؤى /مناظير جديدة، وضرورة وصف دلالات ومعاني البيانات كما تستخدم في مجالها/المؤسسة أو المنظمة المعنية.
وهي طريقة فريدة من أجل تصميم قاعدة بيانات مثالية، رغم أنه يجب الأخذ بالاعتبار أننا في التصميم الموزع، ونحتاج إلى وضع الجداول فعليا على الشبكة، كما نحتاج إلى تقسيم الجداول.
شكل توضيحي لهذه الإستراتيجية ص ٨.

التصميم من القاع إلى القمة Bottom Up:

طريقة التصميم من أعلى لأسفل تكون خيارا جيد إذا كان لديك حرية البدء من الصفر. ولكن للأسف ليست هذه الحالة الدائمة، فحالات التصميم من القاع للقمة هي الأكثر شيوعا.
التصميم من القاع للقمة، يدمج المخططات المستقلة/شبه المستقلة في المخطط الأولي العام (GCS). فيجب التعامل مع قضايا تحويل المخطط، وقد يتعامل مع قضايا التكامل/الدمج غير المتجانس.

تصميم قاعدة البيانات الموزعة Distributed Database Design:

من أجل تبسيط الأشياء، سوف نفترض أن المصممين لديهم سلطة كاملة عند تصميم قاعدة البيانات، ونعلم أن أغلب عمليات التصميم المبدئية، هي فريدة من أجل تصميم قاعدة بيانات وحيدة، ومع ذلك عند تقييم المتطلبات لمختلف المستخدمين والتطبيقات، يكون لدينا وعي باحتياجاتهم، مثل الموقع، ومرات الوصول، ونوع البيانات التي يطلبونها مرارا.
أحيانا يكون من الواضح للمصمم أن جداولا معينة نحتاجها كاملة في مكان محدد، هذه العملية قد تتطلب تقسيم قاعدة البيانات.

تصميم قواعد البيانات الموزعة، تصميم التقسيم – Distributed Database Design Fragmentation

ما هي مميزات التصميم أو لماذا نقوم بالتقسيم؟:
عادة تكون رؤية التطبيق مركزة على مجموعة جزئية العلاقة (الجدول)، سواء كان الجزء أفقيا (الصفوف/السجلات) أو عموديا (الأعمدة/الحقول) Both vertical & horizontal، ومن المعتاد أيضا أن هذه التطبيقات موزعة. ونكون في حاجة إلى أن تكون البيانات كثيرة الوصول من تطبيق معين قريبة إلى ذلك التطبيق، ولهذا نستخدم ما يسمى بالمرجعية المحلية Locality of reference.
المرجعية المحلية تساعدنا على أن تكون البيانات قريبة من التطبيقات التي تعمل عليها دائما، وهذا يفيدنا في ما نسميه بالتنفيذ المتزامن !concurrent execution.

مضاعفة القسم Fragment Replication:

هنا تشابه بين مصطلحي التقسيم والمضاعفة، وأحيانا يعنيان الشيء نفسه، وعلى المصمم أن يناقش ما إذا كان من الصعوبة أن يضاعف/يكرر أقسام معينة، فالاسترجاع يكون أكثر فاعلية إذا كانت البيانات قريبة من التطبيق، ولكن تظهر لنا مشكلة صعوبة التحديث، بسبب ضرورة تحديث نفس البيانات في أماكن مختلفة. كما أن هناك ميزة إن توقف موقع معين، إذا كانت بياناته في موقع آخر. وهذا الموضوع يناقش لاحقا.

عيوب التقسيم أو لماذا لا نقوم بالتقسيم Fragmentation -Why Not

- صعوبة تحديث البيانات الموزعة.
- تأثر الأداء سلبيا.
- تكلفة إضافية متعلقة باختلاف منصات العمل (التغير heterogeneous).

- مشكلة فحص السلامة الدلالية Semantic integrity checking تناقش لاحقا.
- إذا حصل فشل لأحد مواقع التوزيع، يكون على النظام ضمان عدم فقدان التحديثات.

أمثلة على التقسيم Fragmentation Examples:
ليكن لدينا الجدول التالي، جدول مشاريع مؤسسة ما PROJECT table:

PROJ:

PNO	PNAME	BUDGET	LOC
P1	Instrumentation	150000	Montreal
P2	Database Develop.	135000	New York
P3	CAD/CAM	250000	New York
P4	Maintenance	310000	Paris

عينة لتقسيم أفقي Sample Horizontal Fragmentation:
على افتراض كان لدينا مستخدمين في موقعين مختلفين، أحدهم يرى البيانات ذات الميزانية المرتفعة والآخر يرى البيانات ذات الميزانية المنخفضة، نقوم بالتقسيم إذا حسب العمود BUDGET، كالتالي:

$PROJ_1$: (low budgets)

PNO	PNAME	BUDGET	LOC
P1	Instrumentation	150000	Montreal
P2	Database Develop.	135000	New York

$PROJ_2$: (high budgets)

PNO	PNAME	BUDGET	LOC
P3	CAD/CAM	250000	New York
P4	Maintenance	310000	Paris

عينة تقسيم عمودي Sample Vertical Fragmentation
على افتراض ان هناك مستخدمين في موقع معين مهتم بالبيانات المالية للمشاريع، ومستخدمين آخرين مهتم بباقي بيانات المشاريع.

- $PROJ_1$: (financials)

PNO	BUDGET
P1	150000
P2	135000
P3	250000
P4	310000

- $PROJ_2$: (project information)

PNO	PNAME	LOC
P1	Instrumentation	Montreal
P2	Database Develop.	New York
P3	CAD/CAM	New York
P4	Maintenance	Paris

تأدية تقسيم صحيح Achieving Correct Fragmentation
يمكن أن نستخدم أي واحدة من استراتيجيات التقسيم التقليدية، مع ضرورة أن يكون في استطاعتنا إثبات صحتها، وصحة التقسيم تضمن لنا شرطين:
١. حفظ التقسيم للبيانات preserves the data .
٢. عدم ازدواج البيانات duplicate data، بسبب التقسيم.
وعليه فأي إستراتيجية من استراتيجيات التقسيم يجب أن تكون صحيحة بشكل مثبت provably.

قواعد صحة التقسيم Correctness Rules of Fragmentation:

هناك ثلاث قواعد يجب توفرها في التقسيم الحاصل ليكون صحيحا هي:

١. أن يكون التقسيم تاما Completeness (التمام).
٢. أن يكون من الممكن إعادة البناء Reconstruction.
٣. أن يكون التقسيم غير مزدوج (Disjointness).

١. التمامCompleteness .

ليكن لدينا العلاقة (أو الجدول) R ، ولنقسمها إلى العلاقات: R1, R2, ..., Rn، يجب أن تكون جميع عناصر البيانات في العلاقة R موجودة في هذا التقسيم، أي توجد على الأقل في واحده من العلاقات Ri.

وهذا يشبه خاصية في الجبر العلائقي تسمى lossless decomposition property، والمعنى في الواقع هو أن تكون البيانات موجودة كلها، فلا يتم تجاهل أي من عناصرها أثناء التقسيم.

٢. إعادة البناء Reconstruction.

يجب أن يكون من الممكن أن نقوم بتعريف المؤثر (نبلا)، الذي يقوم بإعادة بناء العلاقة R من Ri، وبالطبع فالعملية Ri ستختلف حسب شكل التقسيم، وهذا يعني في الواقع إمكانية إعادة تجميع البيانات.

٣. عدم الازدواج (Disjointness).

هي القاعدة الثالثة من قواعد صحة التقسيم، وتعني بالنسبة للتقسيم الأفقي أن تكون كل السجلات (الصفوف) موجودة في قسم واحد فقط، بمعنى عدم ظهور السجل في أكثر من قسم، فإن أردنا سجلا من قسم آخر علينا جلب القسم كاملا.

وبالنسبة للتقسيم العمودي، تكون كل الأعمدة غير المفتاحية (*non-key*columns) موجودة في قسم واحد، فمن الضروري تكرار المفتاح الرئيسي أو بعض المفاتيح الأخرى، من أجل إعادة البناء.

وكل هذا يعني في الواقع: لا يجب أن يوجد ازدواجية في البيانات *duplicates*. والإزدواجية تختلف عن التكرار/التضاعف في كونها تكرارا غير مقصود وغير مسيطر عليه.

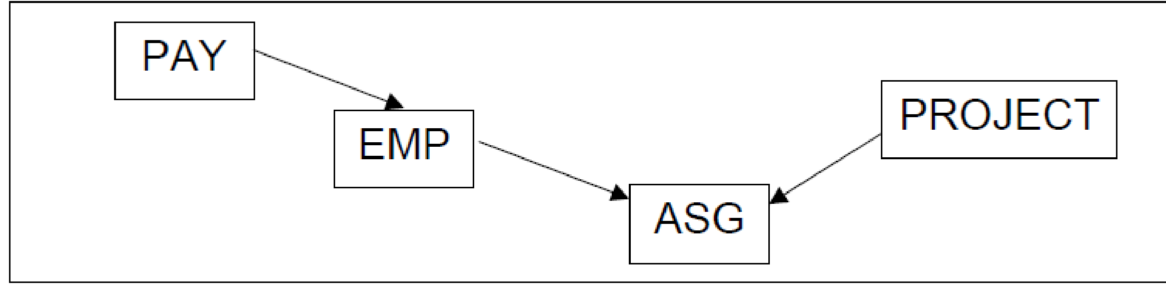
التقسيم الأساسي والتقسيم المشتق Primary vs. Derived Fragmentation

التقسيم الأساسي *Primary* fragmentation:

هو تقسيم يتم اشتقاقه بواسطة آلية اكتشاف heuristics محددة، تطبق على الجدول.

التقسيم المشتق *Derived* fragmentation:

هو تقسيم يتم اشتقاقه اعتمادا على تقسيم سابق لجدول مرجعي، (قد يكون مر بتقسيم آخر أساسي أو مشتق).



مثال: بالنظر إلى الشكل السابق، لدينا:

PAY (primary) vs. EMP (derived)
EMP (primary) vs. ASG (derived)

آلية اكتشاف التقسيم Heuristics for Fragmentation:

عملية التقسيم عملية معقدة، فاحتمال عدد التقسيمات الممكنة، يتزايد أسيا وفق زيادة الاستعلامات البسيطة المحتملة. أو ما يسمى بالمسند predicate وهي الجزء الشرطي في الاستعلام البسيط كما سيأتي.

فإذا كان عدد المسندات يساوي n، فهذا يعني أن عدد التقسيمات المحتملة من الترتيب :

$2^n = O(2^n)$

المسندات Predicates

المسندات Predicates: هي الجزء الرياضي في جملة الشرط WHERE clause من الإستعلام المعروف.

وأهميتها تكمن في كونها تحدد التقسيم.
لتكن R(A1, A2, ..., An) علاقة، فنقول أن p_i مسند بسيط، إذا كانت من الشكل:
piΘValue where Θ ∈ { =, <, >, ≤, ≥, <> }
وكأمثلة على ذلك، المسندات:
–PNAME = ’CAD/CAM’
–BUDGET > 200000
وعادة تكون مجموعة من المسندات ضروية لوصف اختيار الصفوف في العلاقة، وتحول أغلب علاقات التركيب البولية Boolean المنطقية، إلى شكل طبيعي مقترن *conjunctive normal form:*
$p_1 \wedge p_2 \wedge \ldots \wedge p_k$
نحاول عادة تقسيم الجداول باستخدام أنماط الإختيار المحتملة وفق (WHERE clause)، أما تركيب مجموعة من المسندات في الشكل الطبيعي المقترن فتسمى بالـ *minterm.*
وليكن لدينا مجموعة من المسندات في العلاقة كالتالي:
$Pr = \{p_1, p_2, \ldots, p_k\}$

الـ Minterm:
ليكن لدينا مجموعة مسندات الـ minterm التالية:
$M = \{ m_1, m_2, \ldots, m_z\}$
حيث
$M = \{m_j | m_j = \wedge\ p_n\ (p_n \in P_r)\ \}$
مع أخذ الاعتبار للخصائص التالية:
بالنسبة للمساواات يكون: !(attr= val) = (attr<> val) .
وبالنسبة للمتراجحات !(attr> val) = (attr≤val).
ليس من الضروري ازدواج المسندات، وخاصة في الـ minterm الواحدة.

مثال للـ Minterm: باستخدام جدول المشاريع أعلاه، نحصل على المسندات الخمسة التالية:
p1: LOC = ’Montreal’
p2: LOC = ’New York’
p3: LOC = ’Paris’
p4: BUDGET > 200000
p5: BUDGET <= 200000
ومن هذه المسندات يمكن تكوين عدد كبير من الـ Minterm، منها:
m1: LOC = ’New York’ ^ BUDGET > 200000
m2: LOC = ’New York’ ^ BUDGET <= 200000
m3: LOC = ’Paris’ ^ BUDGET > 200000
m4: LOC = ’Paris’ ^ BUDGET <= 200000
m5: LOC = ’Montreal’ ^ BUDGET > 200000
m6: LOC = ’Montreal ’ ^ BUDGET <= 200000

خصائص المينتيرم Minterm Properties:
١. خاصية إختيار المينتريم (sel)
هي عدد الصفوف/السجلات التي تظهر مع كل واحده من المينتيرم، مثلا بالنسبة للمينتريم السابقة واعتمادا على الجدول proj السابق، يكون:
$sel(m_1) = 1; sel(m_2) = 1; sel(m_4) = 0$
كما نستخدم أيضا $card(f_i)$ للدلالة على عدد صفوف التقسيم الناتج عن المينتيرم m_i.
٢. خاصية عدد مرات الوصول للصفوف بواسطة التطبيق او المستخدمين
ليكن لدينا مجموعة الإستعلامات:
$Q = \{q_1, q_2, \ldots, q_q\}$
فيكون acc(q1) هو عدد مرات الوصول للإستعلام الأول query 1.
التقسيم الأساسي الأفقي Primary Horizontal Fragmentation
باستخدام عدد مرات الوصول و المينتيرم يمكن توليد التقسيم الأفقي.

ليكن لدينا عدد التقسيمات w ، عندئذ كل تقسيم معطى للعلاقة R_i باستخدام الصيغة F_i ، حيث أن كل صيغة تمثل تعبير مينيتيرم أو مسند predicate.

$R_i = \sigma_{F_i}(R)$ where $1 <= i <= w$

•Examples:

–PROJ1= $\sigma_{BUDGET<=200000}$ (PROJ)

–PROJ2= $\sigma_{BUDGET>200000}$ (PROJ)

تذكير:

التعبير $\sigma_{BUDGET>200000}$ (PROJ) يستخدم في الجبر العلائقي بمعنى مقابل في لغة SQL، ويعني: (select*PROJ where BUDGET>200000)

ملاحظات متعلقة بالتقسيم الأفقي Horizontal Fragmentation Issues

- يعود إلى مصمم قاعدة البيانات اختيار تصميم التقسيم الصحيح.
- وهذا يعتمد عادة على فهم الإستعلامات والتطبيقات وعدد مرات الوصول بينها.
- لهذا سيتم وصف آلية الكشف heuristic لتحديد التقسيم باختصار.
- مسألة تعديل التقسيم لاحقا تعتمد على المعلومات الجديدة، إضافات بيانات أو تفاصيل تصميم مثل:
  - ماذا لو تم إضافة صف لمشروع جديد فيه BUDGET = 600، هل يؤدي هذا إلى ضرورة إنشاء تقسم جديد، ما هي حدوده، أم انه لا يؤثر على التقسيم الحالي.

تحديد المينتيرم المثالي Optimal Minterm Determination:

يجب ان يتسم كل تقسيم بخاصية الاحتمالية المتساوية لوصول كل تطبيق إلى كل صف فيه. والصفوف في التقسيمات المختلفة يجب أن تكون لها احتماليات مختلفة للوصول من التطبيقات المختلفة. كما نحتاج إلى خاصية التمام ولكن دون تضييع أي سجل، مع الأخذ بالاعتبار وجود صفوف جديدة محتملة. ونريد التقليل minimality بالأقسام ولكن دون أن نفرط بأي تقسيم محتمل. وهذا يعني وجود مسندات predicates مطلوبة لتكون مناسبة للتقسيم، مع المحافظة على الشرط التالي عند اختيار التقسيم:

ليكن لدينا تقسيمان R_i،R_j هما F_i و F_j حيث $F_i=m_i$ و $F_j=m_j$ يجب أن يكون:

acc(mi)/ card(fi) ≠ acc(mj)/ card(fj)

أي بمعنى الوصول للمسند رقم i لا يساوي الوصول للمسند j :

Paccess(i) ≠Paccess(j)

القضية الأساسية هي كيف يجب أن تكون الاحتمالات مختلفة لنقوم بإنشاء تقسيم جديد.

خوارزمية تحديد المينتيرمز Algorithm for Determining Minterms

القاعدة الأولى : يجزء التقسيم إلى جزئين على الاقل بحيث يكون الوصول مختلفا بينهما من تطبيق/او مستخدم واحد على الأقل.

سوف نستخدم في هذه الخوارزمية المتغيرات التالية:

R :relation	العلاقة
Pr :set of simple predicates	مجموعة مسندات إبتدائية
Pr' :another set of simple predicates	مجموعة مسندات أخرى
F :set of minterm fragments	مجموعة التقسيمات

الخطوة الأولى :

قم بإيجاد الحد الأدنى من مجموعة المسندات التامة

Step 1: Find Complete and Minimal Set of Predicates

COM_MIN (R, Pr) {// Compute minterms

// find a pisuch that pipartitions R according to Rule 1

Pr' = pi

Pr = Pr -pi

F = fi// fiis mintermfragment according to pi

while (Pr' is incomplete) {

find pj Pr that partitions some fkof Pr'

```
Pr’ = Pr’ U pj
Pr = Pr -pj
F = F U fj// fjis mintermfragment according to pj
if pkPr’ which is non-relevant{ // this is complex
Pr’ = Pr’ -pk
F = F -fk
}
}
return Pr’
}
```

ملاحظات على إيجاد التقسيمات المناسبة Issues of Finding Relevant Fragments

كيف يمكن لشيء معين ان يكون مناسبا مبدئيا ونرى بعدها أنه غير مناسب ؟

يمكن أن تكون فكرت مبدئيا أن الموقع عامل مهم ومميز، لأن المواقع المختلفة يتم الوصول إليها باستعلامات مختلفة، ولكنك عندما تقوم بالتقسيم حسب الموقع، يرى آخر أن الأهم في هذه الحالة هو الـ budget ، كما راينا في المثال السابق للتقسيم الأفقي، وهو خلاف يحدث كثيرا، لأن كل موقع يتجه لأخذ الميزانيات الأعلى.

هناك إذا قضايا أصعب من أن يفكر فيها المصمم، كوجود مسندات كثيرة يمكن ان تستخدم للتقسيم، أو ضرورة تقديم مبررات معقولة للبدء بمسندات معينة، فيجب على كل الحال استخدام استعلامات مبدئية عند بدء التصميم.

الخطوة الثانية: **أسرد مجموعة المينيتيرم كلها**

Step 2: Define Full Minterm Set

نحتاج لتعريف كل مجموعة المينيتريم اعتمادا على مجموعة المسندات المعطاة سابقا. وهذا يعطي عدد كبير يتزايد اسيا حسب عدد المسندات، مثلا:

$m_1= p_1, m_2= p_2, m_3= p_3, m_4= p_4, m_5= p_5,$
$m_6= p_1 \wedge p_2, m_7 = p_1 \wedge p_3, ..., m_{10}= p_2 \wedge p_3, ...,$
$m1_6= p_1 \wedge p_2 \wedge p_3, m_{17}= p_1 \wedge p_2 \wedge p_4, ...,$
$m_{26}= p_1 \wedge p_2 \wedge p_3 \wedge p_4, ...,$
$m_{31}= p_1 \wedge p_2 \wedge p_3 \wedge p_4 \wedge p_5$

الخطوة الثالثة: حدد الاستدلالات

Step 3: Define Inferences

نعرف مجموعة من الإستدلالات بالإعتماد على المسندات التي لدينا.

مثلا بافتراض لدينا القيمتين val1 و val2 تامتان وتتمم كل واحدة الأخرى، حسب الخطوة الأولى، وكانت المسندات كالتالي:

p_1: att= val1
p_2: att= val2

نستطيع الحصول على الاستدلالين i_1 و i_2 :

i_1: (att= val1) => !(att= val2)
i_2: (att= val2) => !(att= val1)

فتكون مجموعة المينترم الممكنة:

m_1: (att= val1) ^ (att= val2)
m_2: (att= val1) ^ !(att= val2)
m_3: !(att= val1) ^ (att= val2)
m_4: !(att= val1) ^ !(att= val2)

عندئذ بالاستعانة بالاستدلالين اعلاه، نخرج بالاستنتاج التالي: المينتيرم m_1 و m_2 لا يمكن قبولهما بسبب أنهما يناقضان الإستدلالات السابقة، ونقوم بحذفهما من مجموعة المينتيرم.

الخطوة الرابعة: **التخلص من المينتيرم المضمنة أو الجزئية**

Step 4: Eliminate Subsumed Minterms

نتخلص من كل مينتيرم موجود في مينتيرم آخر من مجموعة المينيتيرمز، مثلا:

إذا كان $m_a = p_1, m_b = p_2, m_c = p_1 \wedge p_2$

عندئذ نقوم بحذف m_a و m_b لأنهما موجودان في m_c أي أن m_c □ m_a and m_c □ m_b

وبشكل مشابه ننظر للمثال التالي:

–if: $m_a = p_1 \wedge p_2$, $m_b = p_1 \wedge p_3$, and $m_c = p_2 \wedge p_3$ as well as $m_d = p_1 \wedge p_2 \wedge p_3$

–Then: eliminate m_a, m_b, and m_c because m_d □ m_a , m_d □ m_b and m_d □ m_c

الخطوة الخامسة: نحسب جدول المينيتيرم كمراجعة للخطوات السابقة

Step 5: Calculate Minterms for Table

PHORIZONTAL	التقسيم الأساسي الأفقي
{	}
Pr' = COM_MIN(R, Pr)	المسندات متتامة.
determine set of minterms M	تحديد مجموعة المنيتيرم
determine inference set I among Pr'	تحديد مجموعة الاستدلال وفق Pr'
eliminate contradictory mi's according to I from M	حذف التناقض باستخدامها
eliminate subsumed minterms	التخلص من المينيتيرم الجزئية
what is left in M is horizontal fragmentation	ما تبقى الآن هو التقسيم الأفقي
}	{

فيما يلي مثال كامل ينفذ ما سبق على الجدول PROJ:

Example

Step 1: Identify relevant predicates

p_1: LOC = 'Montreal'

p_2: LOC = 'New York'

p_3: LOC = 'Paris'

p_4: BUDGET > 200000

p_5: BUDGET <= 200000

Define Full Minterm Set

m_1: LOC = 'Montreal'

m_2: LOC = 'New York'

m_3: LOC = 'Paris'

m_4: BUDGET > 200000

m_5: BUDGET <= 200000

m_6: LOC = 'Montreal' ^ LOC = 'New York'

m_7: LOC = 'Montreal' ^ LOC = 'Paris'

m_8: LOC = 'Montreal' ^ BUDGET > 200000

m_9: LOC = 'Montreal' ^ BUDGET <= 200000

m_{10}: LOC = 'New York' ^ LOC = 'Paris'

m_{11}: LOC = 'New York' ^ BUDGET > 200000

m_{12}: LOC = 'New York' ^ BUDGET <= 200000

Define Full Minterm Set (cont.)

m_{13}: LOC = 'Paris' ^ BUDGET > 200000

m_{14}: LOC = 'Paris' ^ BUDGET <= 200000

m_{15}: BUDGET > 200000 ^ BUDGET <= 200000

m_{16}: LOC = 'Montreal' ^ LOC = 'New York' ^ LOC = 'Paris'

m_{17}: LOC = 'Montreal' ^ LOC = 'New York' ^ BUDGET > 200000

m_{18}: LOC = 'Montreal' ^ LOC = 'New York' ^ BUDGET <= 200000

m_{19}: LOC = 'Montreal' ^ LOC = 'Paris' ^ BUDGET > 200000

m_{20}: LOC = 'Montreal' ^ LOC = 'Paris' ^ BUDGET <= 200000

m_{21}: LOC = 'Montreal' ^ BUDGET > 200000 ^ BUDGET <= 200000

m_{22}: LOC = 'New York' ^ LOC = 'Paris' ^ BUDGET > 200000

m_{23}: LOC = 'New York' ^ LOC = 'Paris' ^ BUDGET <= 200000

m_{24}: LOC = 'New York' ^ BUDGET > 200000 ^ BUDGET <= 200000

m_{25}: LOC = 'Paris' ^ BUDGET > 200000 ^ BUDGET <= 200000

Define Full Minterm Set (cont.)

m_{26}: LOC = 'Montreal' ^ LOC = 'New York' ^ LOC = 'Paris' ^
BUDGET > 200000

m_{27}: LOC = 'Montreal' ^ LOC = 'New York' ^ LOC = 'Paris' ^
BUDGET <= 200000

m_{28}: LOC = 'Montreal' ^ LOC = 'New York' ^ BUDGET > 200000 ^
BUDGET <= 200000

m_{29}: LOC = 'Montreal' ^ LOC = 'Paris' ^ BUDGET > 200000 ^
BUDGET <= 200000

m_{30}: LOC = 'New York' ^ LOC = 'Paris' ^ BUDGET > 200000 ^
BUDGET <= 200000

m_{31}: LOC = 'Montreal' ^ LOC = 'New York' ^ LOC = 'Paris' ^
BUDGET > 200000 ^ BUDGET <= 200000

Define Inferences

- Inferences:

  $p_1 \Rightarrow \sim p_2$

  $p_1 \Rightarrow \sim p_3$

  $p_2 \Rightarrow \sim p_1$

  $p_2 \Rightarrow \sim p_3$

  $p_3 \Rightarrow \sim p_1$

  $p_3 \Rightarrow \sim p_2$

  $p_4 \Rightarrow \sim p_5$

  $p_5 \Rightarrow \sim p_4$

- Left with only:

  m_1: LOC = 'Montreal'

  m_2: LOC = 'New York'

  m_3: LOC = 'Paris'

  m_4: BUDGET > 200000

  m_5: BUDGET <= 200000

  m_8: LOC = 'Montreal' ^ BUDGET > 200000

  m_9: LOC = 'Montreal' ^ BUDGET <= 200000

  m_{11}: LOC = 'New York' ^ BUDGET > 200000

  m_{12}: LOC = 'New York' ^ BUDGET <= 200000

  m_{13}: LOC = 'Paris' ^ BUDGET > 200000

  m_{14}: LOC = 'Paris' ^ BUDGET <= 200000

- After subsumption, only **m_8, m_9, m_{11}, m_{12}, m_{13}, m_{14}** remain

Actual Partitions

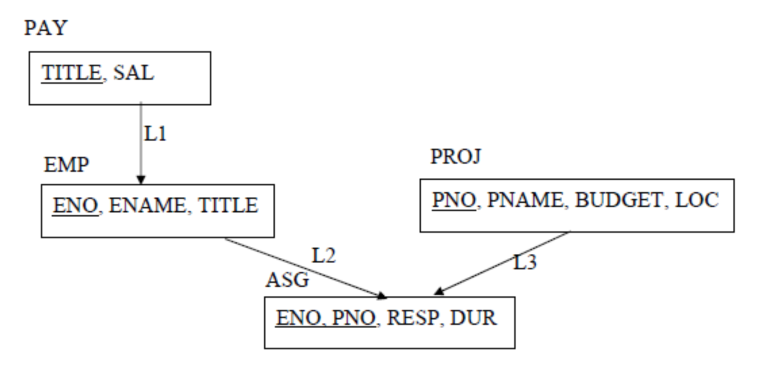
- The four actual partitions are: m_9, m_{11}, m_{12}, m_{13}

PNO	PNAME	BUDGET	LOC
P1	Instrumentation	150000	Montreal
P2	Database Develop.	135000	New York
P3	CAD/CAM	250000	New York
P4	Maintenance	310000	Paris

- The two partitions m_8 and m_{14} have no data

التقسيم الأفقي المشتق Derived Horizontal Fragmentation

في هذا النوع يكون التقسيم بالاعتماد على جدول مشتق سابقا، مثلا في الشكل التالي نقوم بتقسيم الجدول EMP إعتمادا على تقسيم أساسي للجدول PAY.



بإفتراض أن الجدول PAY قد قسم حسب:

PAY1= $\sigma_{salary<=30000}$ (PAY)
PAY2= $\sigma_{salary>30000}$ (PAY)

عندئذ نقوم بتقسيم الجدول EMP بسهولة اعتمادا على تقسيمنا لـ PAY، وذلك باستخدام علاقة الارتباط JOIN التي يرمز لها بالرمز ⋈ في الجبر العلائقي، كالتالي:

–EMP_i= EMP ⋈PAY_i for all i

ويعني ذلك استخدام المفاتيح الأجنبية والرئيسية بين الجدولين لتكوين مسندات تقسيم الجدول EMP كل مره i، وفق أحد تقسيمات PAY المناظرة. ويكون الشرط بالطبع:

EMP.TITLE = PAY.TITLE

هذا التقسيم يعتمد بالطبع على بيانات كل من الجدولين بالطبع، لهذا سمي تقسيما مشتقا.

تقسيم مشتق للجدول Derived Fragmentation for ASG ASG

والآن عندما نقوم بعمل تقسيم للجدول ASG، لدينا جدولين أو علاقتين مرشحتين للتقسيم، بالنظر إلى الشكل أعلاه، هما EMP و PROJ.

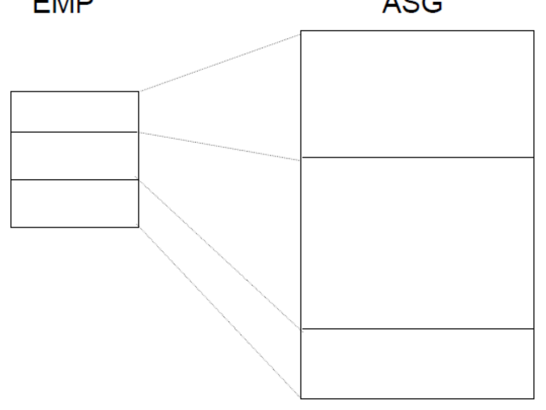
نختار التقسيم الذي نريد حسب:

- خصائص الربط الأحسن بين الجدولين.
- الاستخدام الأكثر في التطبيقات فيجب الأخذ بالاعتبار ان اختيار التقسيم يكون دائما مرتبطا بالاستخدام.

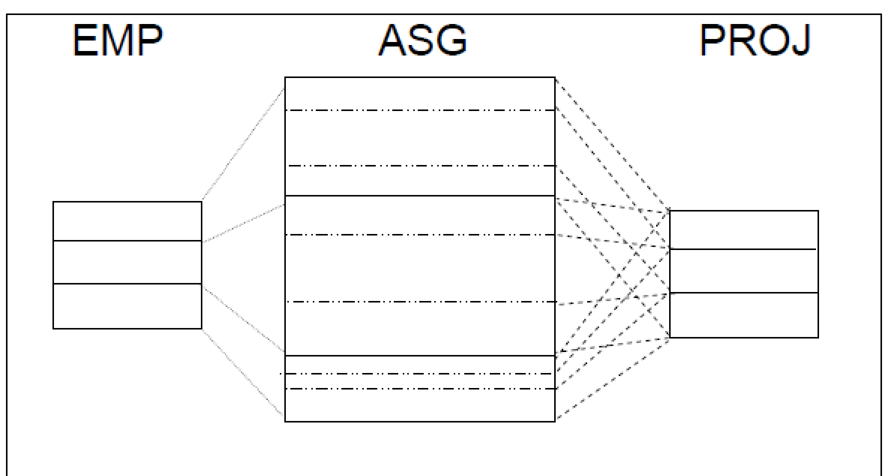
ليس من السهولة أن نقرر فذلك يعتمد على:

- حجم العلاقات الجزئية.
- نوع الإرتباط الموزع.
- الاستعلامات المتوازية، وتوزيع المعالجات عبر النظام، وحجم عمليات الدمج والاتحاد المستخدمة.





لماذا لا نقوم بالتقسيم المشتق من كلا الجدولين في الحالة السابقة :
سيكون الأمر اكثر تعقيدا لو قمنا بذلك، وهذا لا يحقق هدف التقسيم البسيط الذي نحدده دائما.



الخلاصة والاستنتاج Conclusion:

- يعتمد تصميم قاعدة البيانات الموزعة على :

النموذج الأولي للمؤسسة، وعدد مرات ومواقع وصول التطبيق للبيانات.

- جداول البيانات توضع قرب التطبيق الذي يحتاجها أكثر من غيره.
- تقسيم الجدول مفيد في نظم قواعد البيانات الموزعة، عندما يكون للمستخدمين والتطبيقات متطلبات مختلفة للوصول إلى نفس الجدول.
- التقسيم مفيد بإتاحته وصولا سريعا خلال المرجعية المحلية للجدول.
- المعايير التي تتحكم بتقسيم الجدول هي:
    - عدد مرات الوصول.
    - الموقع يتحكم بعملية التقسيم.
    - الاستعلامات والمسندات المشتقة من العلاقات.

* التقسيم الأساسي يتحكم بالتقسيم المشتق.
* يجب أن يكون كلا من التقسيم الأساسي والمشتق مثبتا بصورة صحيحة.

Vertical partitioning

Vertical partitioning is complex
If number of non-key columns = m, the number of possible fragments is the mth Bell number B(m) ≈ mm for large m
This is a big number

Frag(a,b) = a, b | ab B(2) = 2
Frag(a,b,c) = a, Frag(b,c) | ab, c | ac, b | abc B(3) = 5
Frag(a,b,c,d) = a, Frag(b,c,d) | ab, Frag(c,d) |
ac, Frag(b,d) | ad, Frag(b,c) | abc, d |
abd, c | acd, b | abcd B(4) = 15
Frag(a,b,c,d,e) = a, Frag(b,c,d,e) | ab, Frag(c,d,e) |
ac, Frag(b,d,e) | ad, Frag(b,c,e) |
ae, Frag(b,c,d) | abc, Frag(c,d) | abd, Frag(c,e) |
abe, Frag(c,d) | acd, Frag(b,e) | ace, Frag(b,d) |
ade, Frag(b,c) | abcd, e | abce, d | abde, c |
acde, b | abcd B(5) = 52

Sample clustering

q1: SELECT budget FROM proj WHERE pno = value
q2: SELECT pname, budget FROM proj
q3: SELECT pname FROM proj WHERE loc = value
q4: SELECT sum(budget) FROM proj WHERE loc = value

Define a matrix from the use() function

	A1 (pno)	A2 (pname)	A3 (budget)	A4 (loc)
q1	1	0	1	0
q2	0	1	1	0
q3	0	1	0	1
q4	0	0	1	1

This does not help us because we don’t know the access frequency of the attributes
We need the access frequency to calculate attribute affinity between two columns Ai, Aj – aff(Ai, Aj)
The attribute affinity between two attributes Ai and Aj of a relation

∑ (k|use(qk,Ai) & use(qk,Aj)) ∑ (∀PAYl) [refl(qk) accl(qk)]
where:
refl(qk) is the number of access to attributes Ai and Aj for each application qk at site Sl
and

accl(qk) is the access frequency of qk for each site

Suppose 3 sites and refl(qk) = 1 for each site

acc1(q1) = 15 acc2(q1) = 20 acc3(q1) = 10
acc1(q2) = 5 acc2(q2) = 0 acc3(q2) = 0
acc1(q3) = 25 acc2(q3) = 25 acc3(q3) = 25
acc1(q4) = 3 acc2(q4) = 0 acc3(q4) = 0

The attribute affinity matrix

	A1 (pno)	A2 (pname)	A3 (budget)	A4 (loc)
A1	45	0	45	0
A2	0	80	5	75
A3	45	5	53	3
A4	0	75	3	79

aff(A2, A2) = acc1(q2) + acc2(q2) + acc3(q2) + acc1(q3) + acc2(q3) + acc3(q3) = 80
aff(A2, A4) = acc1(q3) + acc2(q3) + acc3(q3) = 75
etc.

Clustering Algorithm
For small case, can examine the matrix

	A1 (pno)	A3 (budget)	A2 (pname)	A4 (loc)
A1	45	45	0	0
A3	45	53	5	3
A2	0	5	80	75
A4	0	3	75	79

Bond Energy Algorithm for global affinity measure:

AM = $\sum$ (i=1 to n) $\sum$ (j=1 to n)
aff(Ai, Aj) [aff(Ai-1, Aj) + aff(Ai+1, Aj) + aff(Ai, Aj-1) + aff(Ai, Aj+1)]

aff(A0, Aj) = aff(Ai, A0) = aff(An+1, Aj) = aff(Ai, An+1) = 0 // boundary

Since the algorithm is symmetric, we can do it in two pieces - horizontal & vertical

AM horizontal = $\sum$ (i=1 to n) $\sum$ (j=1 to n) aff(Ai, Aj) [aff(Ai, Aj-1) + aff(Ai, Aj+1)]

AM vertical = $\sum$ (i=1 to n) $\sum$ (j=1 to n) aff(Ai, Aj) [aff(Ai-1, Aj) + aff(Ai+1, Aj)]

Algorithm
Initialize - pick any column

Iterate
Select the next column and try to place it in the matrix
Choose the place that maximizes the contribution or global affinity – contribution function is cont()
Repeat the process for rows. However, since AA is symmetric, just reorder the rows!

```
Bond Energy Algorithm (BEA) {

        input: AA
        output: CA

                // put the first two columns in
                CA(*, 1) <- AA(*, 1)
                CA(*, 2) <- AA(*, 2)

// for each of the rest of the columns of AA,
// choose the best placement
        while (index <= n ) {
                // calculate continuity of each
// possible place for new column
                for (i=1; i< index; i++) {
                        calculate cont(Ai-1, Aindex, Ai);
                }
                calculate cont(Aindex-1, Aindex, Aindex+1);  // boundary

                loc <- placement given by maximum cont()
                for (j=index; j>loc; j--) {
                        CA(*,j) <- CA (*, j-1);
                        }
                CA(*, loc) = AA(*, index);

                index <- index + 1;
        }

        // reorder the rows according to the placement of columns
}
```

We can simplify the calculation for the total bond energy.
Remember

$$AM = \sum (i=1 \text{ to } n) \sum (j=1 \text{ to } n)\ aff(Ai, Aj)\ [aff(Ai, Aj-1) + aff(Ai, Aj+1)\ ]$$

which is:

$$AM = \sum (i=1 \text{ to } n) \sum (j=1 \text{ to } n)\ [aff(Ai, Aj)\ aff(Ai, Aj-1) + aff(Ai, Aj)\ aff(Ai, Aj+1)\ ]$$

$$AM = \sum (j=1 \text{ to } n)\ [\quad \sum (i=1 \text{ to } n)\ aff(Ai, Aj)\ aff(Ai, Aj-1) + \sum (i=1 \text{ to } n)\ aff(Ai, Aj)\ aff(Ai, Aj+1)\ ]$$

Define bond() energy between Ax and Ay

bond (Ax, Ay) = ∑ (z=1 to n) aff(Az, Ax) aff(Az, Ay)

AM is rewritten as:

AM = ∑ (j=1 to n) [bond(Aj, Aj-1) + bond(Aj, Aj+1)]

Consider a matrix of

A1A2...Ai-1AiAjAj+1...An
-------------- ----------
AM' AM"

AMold = AM' + AM" +
bond (Ai-1,Ai) + bond (Ai,Aj) + bond (Aj,Ai) + bond (Aj,Aj+1)
= ∑ (l=1 to i-1) [bond(Al, Al-1) + bond(Al, Al+1)] +
∑ (l=1 to i+2) [bond(Al, Al-1) + bond(Al, Al+1)] +
2bond(Ai,Aj)

Consider placing Ak between Ai and Aj

AMnew = AM' + AM" +
bond (Ai-1,Ai) + bond (Ai,Ak) + bond (Ak,Ai) +
bond (Ak,Aj) + bond (Aj,Ak) +bond (Aj,Aj+1)
= ∑ (l=1 to i-1) [bond(Al, Al-1) + bond(Al, Al+1)] +
∑ (l=1 to i+2) [bond(Al, Al-1) + bond(Al, Al+1)] +
2 bond (Ai,Ak) + 2 bond (Ak,Aj)

So cont() is:

cont(Ai, Ak, Aj) = AMnew - AMold
= (∑'s cancel) +
2 bond (Ai,Ak) + 2 bond (Ak,Aj) - 2bond(Ai,Aj)

Partitioning Algorithm
We can eyeball the result matrix to determine the best places to perform vertical fragmentation
However, we would like to do it algorithmically
It may be more accurate
Can't always eyeball the results

	A1A2 ... Aj	Aj ... An
A1A2 ... Aj	TA	
Aj ... An		BA

Examine the set of queries Q = {q1, q2, ... , qq }
Set the partition to ensure that most of the queries access either one or the other partition

AQ(qi) = { Aj | use(qi, Aj) = 1}
TQ = { qi | AQ(qi) ⊆ TA }

BQ = { qi | AQ(qi) ⊆ BA }
OQ = Q - { TQ-BQ }

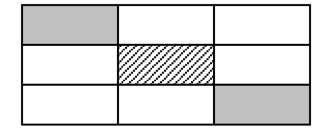
Next define counts of queries

CQ = ∑ (qi∈Q) ∑(∀Sj) refj(qi) accj(qi)
CTQ = ∑ (qi∈TQ) ∑(∀Sj) refj(qi) accj(qi)
CBQ = ∑ (qi∈BQ) ∑(∀Sj) refj(qi) accj(qi)
COQ = ∑ (qi∈OQ) ∑(∀Sj) refj(qi) accj(qi)

Determine the best cut-off point z
There are n-1 choices
The best partition is maximized at:

z = CTQ*CBQ - COQ2

Unfortunately, the following could be the best partitioning:



Need to shift one row at a time and rerun the partitioning algorithm
Prove correctness
Completeness

A (set of attributes of Relation) = ∪ Ri

Reconstruction

FR = {R1, R2, R3, ...}   // set of fragments
R = ⋈K Ri : ∀ Ri ∈ FR, K = key

Distjointness -- true, modulo the keys or the tuple identifier

fhdalqasem@yahoo.com فهد آل قاسم

مقدمة Introduction:

جميع قواعد البيانات، المركزية والموزعة، تحتاج إلى تحكم بيانات معنوي، وهي شروط معينة للبيانات يتم ضبطها من قبل نظام إدارة قاعدة البيانات. وهذا يعني أن المستخدمين سينجزون العمليات الصحيحة التي تتناسب مع الصلاحيات المخولة لهم authorized:

- يستطيع المستخدم فقط إدخال، تحدث و حذف البيانات حسب صلاحياته في قاعدة البيانات.
- يمكن للمستخدم أن يستعلم فقط عن البيانات التي يحق له الوصول إليها.
- للمستخدمين القدرة فقط على القيام بالأشياء الصحية فقط!.

التحكم من أجل الأمنية

يمكن أنجاز أمنية البيانات بثلاثة طرق: هي **المناظير** Views، **و قيود الأمنية** Security constraints، **و قيود السلامة المعنوية** Semantic integrity constraints.

إن انتهاك واحد أو أكثر من هذه الضوابط، يؤدي مباشرة إلى رفض تنفيذ العملية. وعموما، فالصلاحيات أو الامتيازات privileges هذه تعطى من قبل مدير قاعدة البيانات DBA . كما سنوضح فيما يلي.

الإجراء المطبقة Procedure:

في البدء نراجع الحلول المستخدمة في قواعد البيانات المركزية (غير الموزعة)، و من ثم نناقش بعض التعقيدات التي نراها في قاعدة البيانات الموزعة.

يجب أن نخزن قواعد الأمنية وإجراءاتها في كتالوج قاعدة البيانات، وعندما نقوم بتوزيع كتالوج قاعدة البيانات، وهنا تظهر كثير من التعقيدات، والسؤال في هذه الحالة، في أي الحالات نسمح بازدواج تقديم هذه القواعد ومتى نوزعها فقط بدون ازدواج.

إدارة المنظور View Management:

تعريف المنظور View:

المنظور هو جدول (علاقة)افتراضي virtual relation، فلا يتم تجسيده تماما كما هي الجداول الرئيسية، ولكن بيانات المنظور تأتي دائما من جداول رئيسية، ويعتبر نافذة لرؤية المستخدم إلى قاعدة البيانات، حتى لو كانت رؤية محدودة من أجل الأمنية، فالمخطط الخارجي External schemas يمكن أن يكون أنواعا مختلفة من المناظير إلى قاعدة البيانات.

وتستخدم المناظير أساسا لتنفيذ أمنية البيانات، فهي من جهة تساعدة على إخفاء المعلومات hide information، وذلك بعرض فقط المعلومات المخول للمستخدم مشاهدتها أو الاستعلام عنها.

المناظير في قاعدة البيانات المركزية Views in Centralized Database:

المثال التالي ينشئ منظورا اسمه SYSAN، يساعد المستخدم فقط على رؤية رقم الموظف وإسمه إذا كان الموظف يعمل فقط كمحلل نظام.

```
CREATE VIEW SYSAN (eno, ename) AS SELECT eno , ename FROM emp
WHERE title = ’System Analyst’ ;
```

ENO	ENAME
E2	M. Smith
E5	B. Casey
E8	J. Jones

ويمكن التعامل مع المنظور كأنه جدول View is Treated as a Table

لو كان للشخص الذي يستخدم الجدول emp يملك فقط صلاحية الوصول إلى الموظفين الذي يعملون كمحللي نظام، فسوف يعطي امتيازا على المنظور sysan، وليس على الجدول emp.

ومع هذا فالمنظور SYSAN يمكن أن يستخدم مثل أي جدول ويستدعى في اي استعلام، مثلا:

```
SELECT ename, pno, resp FROM sysan, asg WHERE sysan.eno= asg.eno
```

وتسمى عملية تحويل استعلام معبر عنه بالمنظور، إلى استعلام معبر عنه بالجدول بتكييف الإستعلام query modification.

تكييف **الاستعلام** Query Modification

هذه العملية تنفذ بواسطة معالج الاستعلام query processor ، فالاستعلام السابق يحول داخليا إلى الاستعلام:

SELECT ename, pno, resp FROM emp, asg WHERE emp.eno= asg.eno AND title = ’System Analyst’;

المناظير الديناميكية Views

هو منظور ينفذ ويعطي نتائج مختلفة حسب معيار معين، فالمنظور التالي ESNAME تختلف نتيجة تنفيذه حسب المستخدم الذي ينفذه، وذلك حسب القيمة التي ترجعها الدالة USER.

CREATE VIEW ESNAME AS SELECT E2.eno, E2.ename, E2.title FROM emp E1, emp E2 WHERE E1.title = E2.title AND E1.eno = USER

وعلى هذا، فالاستعلام التالي سوف يعرض جميع الموظفين الذين يعملون في القسم الذي يعمل به المستخدم الذي يستعرض الاستعلام.

تحديث المنظور View Updates:

هناك شروط ضرورية لكي يسمح نظام إدارة قاعدة البيانات بتحديث المنظور وذلك بإضافة البيانات المدخلة أو المحذوفة المعدلة إلى الجدول الأساسي، ومن هذه الشروط:

يجب أن يشمل المنظور على جميع المفاتيح الرئيسية والأجنبية Primary and foreign keys ، وكذلك على الأعمدة التي يشترط أن لا تكون فارغة non-null columns. مثلا عند الإدخال إلى الجدول emp عن طريق المنظور SYSAN يجعل العمود titile فارغا، لأن العمود titile، ليس من الأعمدة في المنظور.

مثال آخر:

لو كان المفتاح الرئيسي للجدول في المنظور التالي هو العمود eno، فالإدخال إليه عن طريق المنظور يكون متعذرا:

CREATE VIEW VIEW1 AS SELECT ename, title FROM emp;

إدارة **المنظور في قواعد البيانات الموزعة**

View Management in Distributed DBs

إدارة المنظور في قواعد البيانات الموزعة من القضايا المهمة، وتظهر عندئذ عدة إشكالات، خاصة أنه يصعب أن تكون المناظير معرفة حسب أقسام fragments قاعدة البيانات، كما أنه يمكن أن تعامل المناظير كأي جدول في قاعدة البيانات الموزعة، وهذا يقتضي تخزينها في كتالوج قاعدة البيانات بنفس الطريقة.

وبخلاف ذلك، تظهر بعض إشكالات التوزيع، مثلا في أي المواقع (الموزعة) يتم تخزين تعريفات المنظور view definitions؟، عادة يتم تخزينها في بعض المواقع التي تحتوي على التقسيمات العائدة إليها.

وهذا جيد في حالة كون المناظير متوافقة مع التقسيمات المصممة، ولكن في الحالة المعاكسة تصبح المسألة أكثر صعوبة، حيث يجب تعريفها في واحد على الأقل من تلك المواقع.

يقترح البعض أن تكون المناظير مبسطة لكي تتوافق مع التقسيمات المصممة، بحيث نستخدم الاستعلام الواحد من أكثر من منظور مثلا، وبحيث يتكرر المنظور مع التقسيم في حالة تكرر التقسيم.

اما الخيار الأخير، فيقترح ازدواجية Duplicate المناظير في جميع المواقع، رغم التكلفة الكبيرة للصيانة والتحديث في هذه الحالة، ورغم بقاء مشكلة متعلقة بالتقسيمات التي تعود لها المناظير، ولهذا يجب أن يكون ازدواج المناظير جزئيا.

وفي كل الحالات السابقة، فإن الاستعلام من المنظور يظل خاضعا لعملية تكييف الاستعلام query modification، الموضحة أعلاه، من أجل دمج بيانات القيود والجداول عند جلب المنظور.

إن الوصول إلى المناظير في الجداول الموزعة مكلف نوعا، فهو يحتاج إلى مساعدة ما قبل الترجمة Pre-compilation ، ويمكن عمله كلقطات Snapshots مخزنة من الجداول، كما يحدث في مستودعات البيانات data warehouse، ويعيب ذلك ضرورة عدم الوصول إلى المناظير إلا في أوقات محددة، تكون فيها اللقطات محدثة، كما أنه يجب في هذه الحالة أن يكون المستخدمين موافقين على احتمال كون البيانات تقريبا محدثة، وليس تماما.

أمنية البيانات Data Security

من أساليب دعم أمنية البيانات أيضا، مسألة الحماية بالتحكم بالامتيازات *authorization control* المصرح بها لكل مستخدم، تقوم حماية البيانات بمنع أي وصول غير مصرح به إلى البيانات، وذلك بتشفير(تعمية) البيانات باستخدام أساليب التشفير المعروفة، كالطريقة القديمة للمفتاح الخاص DES، وطريقة التشفير بالمفتاح العام public-key.

يتم التحكم بالامتيازات بطريقة مشابهة لطريقة نظام التشغيل OS في التحكم بامتيازات بنظام الملفات، حيث أنه في النظم الموزعة distributed DBMS تقوم عدة من المواقع بحماية البيانات من الوصول غير المصرح به لعدد من المستخدمين، من المواقع الأخرى أو من الموقع نفسه.

١. أسلوب النظم المركزية Centralized Systems Approach:

تقوم النظم غير الموزعة أولا بالتحكم بالوصول إلى صفوف البيانات tuples حسب المعيار (user, operation, object) ، والمعيار (مستخدم، عملية، كائن) يقصد به، التصريح لمستخدم معين بتنفيذ عملية معينة على كائن معين، هذا الكائن يكون جدولا أو منظورا أو غير ذلك، كما أن تعريف المستخدم إلى النظام يكون بالمعيار (user, password)، بحيث لا يكفي اسم المستخدم وإنما ضرورة التأكد من كلمة المرور، والكائنات في نظام التشغيل هي الملفات فقط، أما في قاعدة البيانات، فهي كما أسلفنا الجدول أو قد تكون العلاقة، العمود ، أو الصف، كما أنه يمكن لمدير قاعدة البيانات تحديد أي كائنات إضافية.

سياسات منح وحجب الامتياز Grant and Revoke Policies:

الصيغة المستخدمة SQL syntax هي:

GRANT <privilege> ON <object> TO <user>
REVOKE <privilege> FROM <object> TO <user>

حيث أن:

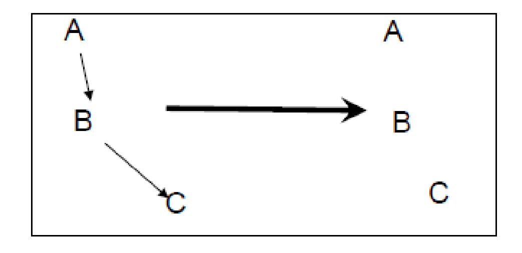
<privilege> = insert, update, delete, select, grant
<user> = public, any user name

وتخزن هذه التفاصيل في قاموس البيانات، مع إضافة المستخدم الذي منح كل صلاحية، وفي الغالب يكون مدير قاعدة البيانات DBA هو الذي يمنح الامتيازات لجميع المستخدمين، وهذا لا ينفي قدرة أي مستخدم على منح امتيازا له امتياز منحه <privilege> = grant لأي مستخدم آخر. فعند منح المستخدم امتيازا لا يملكه هو لا ينفذ، وكذلك يستطيع المستخدم حجب الامتياز الذي منحه هو، ولا يستطيع بالطبع حجب ما لم يمنح.

تتالي حجب الامتياز Cascading Revocation:

هناك حالات كثيرة توضح تعقيد حجب الامتياز، ويمكن توضيح هذا بالمثال الرسومي التالي:

- A grants privileges to B (for some object), who grants privileges to C
- A revokes privileges from B, which cascades to revoking privileges from C



٢. ما يخص **قواعد البيانات الموزعة** Distributed DB Issues

هناك مشاكل إضافية مختلفة عن التعامل مع المناظير ومجموعات المستخدمين، منها مسألة التحقق من هوية المستخدم عن بعد Remote user authentication للموقع الموزع، وكذلك مسألة إدارة قواعد الصلاحيات والامتيازات الموزعة.

بخصوص التحقق عن بعد Remote authentication، يمكن وضع قائمة بجميع المستخدمين وكلمات مرورهم في جميع المواقع الموزعة. مما يجعل دليل إدارة المستخدمين Directory management مسألة مكلفة نوعا ما، حيث يجب أن يكون كل موقع محدثا بجميع المستخدمين، فيستطيع كل مستخدم الدخول من وإلى أي موقع، على أن يقوم كل موقع بصيانة والحفاظ على المستخدمين المحليين لديه local users، متواصلا مع المواقع البعيدة الأخرى، وهذا يجعل إدارة المستخدمين أكثر كفاءة.

إدارة قواعد التحقق من الهوية Authorization Rules
هناك عدة خيارات متاحة منها، التكرار الكامل Fully replicated، حيث يستطاع معالجة قواعد التحقق في كل مرة يتم فيها تكييف الاستعلام query modification، فيكون التأكد من الهوية عند معالجة تنفيذ الاستعلام نفسه.
الخيار الآخر هو التوزيع Distributed، حيث يتم توزيع قواعد التحقق مع التقسيمات الموزعة، وهذا يكون مكلفا في الصيانة Costly to maintain، ولكنه جيد إذا كانت تقنية المرجعية المحلية ممتازة، ويتميز خيار التوزيع بعد ضرورة ترجمة التحقق مع الاستعلام.
ويمكن التعامل مع المناظير وكأنها جداول Views are treated like relations، وفي هذه الحالة تكون إدارة المناظير سهلة، خاصة عند تكرار قواعد التحقق من الصلاحية، كما في الخيار الأول، أما في الحالة الثانية، التوزيع، تكون إدارة المناظير صعبة، خاصة لو كانت المناظير نفسها معرفة في الأقسام الموزعة.
ولتسهيل العملية تخصص المناظير لكل موقع على حده، ويعرف المنظور حسب موقع معين، وتكون لدينا أيضا مناظير عامة، أما المنظور المحلي فيمكن أن يستدعى من الموقع البعيد باستخدام اسم المنظور واسم الموقع مثلا view1@site1، تعني المنظور view1 في الموقع site1، وهذا يساعد على تحقق الصلاحيات وفق تحقق المجموعات groups authorized، حيث تصنف المناظير حسب المواقع.

التحكم بالسلامة المعنوية Semantic Integrity Control
كيف تضمن اتساقية/تناسقية قاعدة البيانات؟، في قاعدة البيانات غير الموزعة تكون الاتساقية اكثر سهولة، ولكن كيف نضمن اتساقية قواعد البيانات الموزعة، فلا يؤدي أي خطأ في التصميم أو التنفيذ إلى ضياع أجزاء منها، أن تناقضها.
نقول أن قاعدة البيانات متناسقة إذا كانت ملبية لشروط السلامة المعنوية، فشروط السلامة المعنوية semantic integrity constraints هي معرفة عن مدى ضمان تناسق قاعدة البيانات، بحيث لا نسمح بتأثر قاعدة البيانات جراء أي حالة عدم تناسق تظهر لنا، ويقدم لنا نموذج البيانات وصفا لبعض هذه القواعد والشروط، من خلال العلاقات بين جداول قاعدة البيانات.
لدنيا نوعين من شروط السلامة المعنوية، أولها تركيبي Structural يعتمد على نماذج البيانات، والثاني سلوكي Behavioral يهتم بما يجب أن تفعله قاعدة البيانات، في الشرط التركيبي نحن نوصف المفاتيح الرئيسية والأجنبية والعلاقات خاصة تلك من نوع One-to-many relationships.
وفي الشرط السلوكي نحن نضبط ردود فعل قاعدة البيانات حيال الأحداث والعمليات الممكنة، وهناك صعوبة في معالجته في بعض الحالات.
حلول متعلقة بإدارة قواعد البيانات المركزية Central DBMS Solutions
شروط السلامة هي توكيدات assertions تتحكم بصفوف البيانات tuple، وفق قواعد الجبر العلائقي، فهي قد تستخدم المقياس العام لكل (∀) للتعبير عن شرط معين يطبق على جميع الصفوف، وقد تستخدم المقياس الخاص (∃) يوجد أو لبعض، الذي يعبر عن شرط يطبق على صفوف معينة.
والتوكيد assert يستخدم في لغة SQL للتعبير عن إجراء يستخدم بدون حدث event ، حيث القادحات triggers تعبر عن إجراءات تعتمد على حدث معين لكي تنفذ، والمثال التالي يوضح كود الـ assert في لغة الاستعلام العالمية المعروفة:

```
CREAT ASSERTION SALARY_CONSTRAINT
CHECK (NOT EXISTS (
SELECT * FROM EMPLOYEE E, EMPLOYEE M, DEPARTMENT D
WHERE E.SALARY > M.SALARY AND E.DNO=D.NUMBER AND D.MGRSSN=M.SSN));
```

أمثلة لشروط السلامة Integrity Constraints
الشرط Non-null:
لا يسمح لعمود معين في الجدول أن يكون فارغا، ENO not null in EMP.
شرط المفتاح الفريد Unique key:
وهو شرط المفتاح الرئيسي أن تكون قيمه فريدة أي غير مكررة، (ENO, PNO) unique in ASG.
شرط المفتاح الأجنبي Foreign key:
هو مفتاح يعود إلى مفتاح رئيسي في جدول آخر، PNO in ASG reference PNO in PROJ.
شرط الاعتمادية الوظيفية Functional dependency:
حيث يقوم عمود معين بتحديد أعمدة معينة في الجدول، ENO in EMP determines ENAME.

نستخدم القادحات كما اشرنا سابقا، للتأكد من تحديث الأعمدة، وذلك باستخدام الأمرين NEW and OLD كما يوضح الشكل العام لها :

–CHECK ON <relation> WHEN <update type> (qualification)

شرط النطاق Domain Constraint:

حيث يتم التأكد هنا من كون عمود معين ضمن مجال محدد من القيم، مثلا:

–CHECK ON PROJ (BUDGET >= 500K & BUDGET <= 1M)

شرط النطاق عند الحذف Domain Constraint on Deletion:

عندما يتم حذف قيمة معينة، يتم ضبطها إلى قيمة تلقائية من اجل تحقيق شرط النطاق، فلا تصير القيمة خارج النطاق كان تكون NULL أو غيرها، مثلا:

–CHECK ON PROJ WHEN DELETE (BUDGET = 0)

التحكم بالتعدي Transition control:

نقوم بضبط قيمة معينة لنضمن عدم تعديها لحد معين، المثال التالي يضبط القيمة الجديدة فلا تكون اقل من السابقة بالنسبة للميزانية، أو مساوية للسابقة بالنسبة للرقم المشروع:

–CHECK ON PROJ (NEW.BUDGET>OLD.BUDGET AND NEW.PNO = OLD.PNO)

فرض السلامة Enforcing Integrity بعد فحص عدم الاتساق Detection of Inconsistencies:

عندما يتم مخالفة شرط معين من الشروط السابقة للسلامة، تكون نتيجة الفحص بالأمر المنطقي CHECK مساوية للقيمة المنطقية false، عندئذ يتم رفض العملية كلها في الغالب، وهذا يسمى بالفحص البعدي posttest، مثلا أمر التحديث Update u: D -> Du، إذا كان أحد شروط السلامة خاطئا، يتم إما تعديل القيمة u أو التراجع عنها، وهذا يعني عدة عمليات، التأكد ثم التراجع.

منع أو إعاقة عدم الاتساق Prevention of Inconsistencies:

يتم السماح بالتحديث فقط عندما تكون قاعدة البيانات في الحالة المتسقة، وهي تسمى بالفحص القبلي pretests، وهي أكثر كفاءة لأنه لا تضطر النظام إلا التراجع.

مثال:

إذا كان لدينا الجملة التالية:

UPDATE PROJ SET BUDGET = BUDGET * 1.1
WHERE PNAME = 'CAD/CAM';

وكان شرط السلامة هو:

CHECK ON PROJ (BUDGET >= 500K & BUDGET <= 1M)

عندئذ تصير جملة الاستعلام أعلاه وكأنها:

UPDATE PROJ SET BUDGET = BUDGET * 1.1
WHERE PNAME = 'CAD/CAM' *AND NEW.BUDGET >= 500000 AND NEW.BUDGET <= 1000000*

وطريقة منع عدم الاتساق Inconsistency Prevention تعمل فقط في حالة صيغة الجبر العلائقي على صفوف الجدول في المقياس العام، والتي تأخذ الأشكال التالية:

–(□x □R) F(x)   // x is the only free variable
–(□x □R) (F(x) => update(x))   // update of R
–(□x □R) (Q(x) & F(x) => update(x))   // update of R with checks

وهي لا تعمل في حالات معينة مثل توكيد المفتاح الأجنبي

–(□g □ASG) (□j □PROJ) : g.PNO = j.PNO

وهناك حلول ضعيفة لهذه المشكلة منها:

بناء اختبار أو فحص قبلي pretests وقت المعالجة compile time، ثم فحص وقت التنفيذ، مع شرط العمل مع جملة تحديث وحيدة على جدول وحيد، مع ملاحظة كثرة الضوابط هنا.

الجداول المتباينة Differential Relations:
كل عملية تحديث على جدول ما ليكن R تنتج ثلاثة جداول جزئية R, R+ و R ، وتكون العلاقتان أو الجدولين الجزئيين R+وR- هما الجدولين المتباينين، فالجدول R+ هو مجموعة الصفوف الجديدة التي تضاف بالعملية INSERT أو UPDATE ، وفي حالة DELETE يكون فارغا من هذه الصفوف.
أما الجدول R- الآخر، فهو مجموعة الصفوف التي حذفت بالعملية DELETE أو UPDATE، في حالة INSERT يكون فارغا.
وعلي هذا يكون لدينا:

•R = R+ ∪ (R -R-) if update

ترجمة التوكيدات Compiled Assertions:
لدينا في عملية التوكيد ثلاثي يتكون من العلاقة R ونوع التحديث T وامر التوكيد نفسه C:
•Triple (R, T, C)
–R -relation
–T -type of update
–C –assertion
يتم فقط اختبار جزء من التوكيدات، التي تتعلق بنفس العلاقة ونفس العملية (matching R & T)، ويتم التطبيق فقط على الجداول (العلاقات) المتباينة فقط R+ و R- .
مثال:
(ASG, INSERT, C1), (PROJ, DELETE, C2), and (PROJ, MODIFY, C3)

C1: (∀NEW ∈ASG+) (∃j ∈PROJ) : NEW.PNO = j.PNO
C2: (∀g ∈ASG) (∀OLD ∈PROJ-) : OLD.PNO ≠g.PNO
C3: (∀g ∈ASG) (∀OLD ∈PROJ-) (∃NEW ∈PROJ+): OLD.PNO ≠g.PNO or OLD.PNO = NEW.PNO

التحكم بالسلامة المعنوية الموزع Distributed Semantic Integrity Control
تعريف توكيدات السلامة Definition of integrity assertions:
هي صفوف محددة بشرط وفق الجبر العلائقي، وتكون إما true أو false ، وتطبق عليها استراتيجات مختلفة حسب نوع التوكيد، منها:
التوكيدات المفردة Individual assertions: عندما يكون الجدول وحيد والمتغير وحيد.
التوكيدات الموجهة للمجموعات Set-oriented assertions:
عندما يكون هناك جدول وحيد ولكن الشروط على متغيرات كثيرة.
التوكيدات الإجمالية Aggregate assertions:
هي أعقد مما سبق وتحتاج إلى معالجة خاصة.
ولنتذكر ان الجداول في القواعد الموزعة تكون مقسمة، ولهذا تكون التوكيدات مقسمة أيضا.

التوكيدات المفردة Individual Assertions
ترسل التوكيدات إلى كل موقع مع التقسيم الموزع نفسه، والفحص والتأكد يتم حيال المسند predicate الذي يحدد التقسيم وحسب البيانات التي فيه، وعندما يتم يظهر لنا توكيد معين للتنفيذ، يتم الفحص حيال جميع المواقع، بحيث:
If assertion(C)=true => p=false at one site, we globally reject predicate
إذا كان التوكيد صحيحا وكان المسند خاطئا في موقع واحد، يتم تعميم رفض نفس المسند في جميع المواقع.
if assertion(C)=true => data does not match, we globally reject predicate
إذا كان التوكيد محققا وكانت البيانات غير مطابقة، يتم تعميم رفض المسند في جميع المواقع.
مثال:

p1: 0 <= ENO < 300
p2: 300 <= ENO <= 600
p3: ENO > 600
C: ENO < 400
حسب المعطيات أعلاه، نلاحظ أن المسند الأول p1 مناسب للتوكيد، وكذلك يمكن أن يكون p2 مناسبا في حالات معينة، أما المسند الثالث فيكون مرفوضا.
fine with p1, OK with p2, if data supports it, not fine with p3 => rejected.

التوكيد الموجه للمجموعة

Set-Oriented Assertion

في حالة وجود عدة جداول، قد تحتوي على مسندات ربط join predicates بين هذه الجداول، ولكننا نعلم أن التوكيدات لا تخزن إلا مع جدول وحيد، لهذا ترسل التوكيدات هذه إلى جميع المواقع التي تخزن تقسيمات ذلك الجدول relation fragments.

يستحيل الفحص اعتمادا على مسندات التقسيمات، لهذا يتم الفحص على البيانات مباشرة، ويكون الفحص أيضا حسب الترابط الحادث بالفعل، وتكون لدينا ثلاث حالات:

١. تقسيمات تكون معتمدة على الربط النصفي semi-joins وفق عمود معين مستخدم في مسند ربط التوكيد.
٢. تقسيمات تكون موجودة في مسند الربط.
٣. تقسيمات غير موجودة في المسند.

مثال:

لدينا التوكيد التالي:

(ASG, INSERT, C1)
where
C1: (□NEW □ASG+) (□j □PROJ) : NEW.PNO = j.PNO

واضح أن التوكيد أعلاه يشترط عدم إدخال صف إلى الجدول ASG دون التأكد من كون المفتاح الأجنبي PNO موجود مسبقا في الجدول الأب PROJ وعليه فإنه مهما كان المسند X الذي يقسم ASG:

•ASG | $X_{PNO}$$PROJ_j$ where $PROJ_j$ is fragment of PROJ

فإنه يجب ان يتم التأكد من جميع التقسيمات $PROJ_j$ الذي في نفس الموقع، وهذا يعني أن معالجة الأمر:

•NEW.PNO = j.PNO

تكون في نفس موقع التقسيمات، فلا نحتاج عندئذ إلى إتصال no communication إضافي.

فإذا تم تقسيم الجدول PROJ أفقيا باستخدام المسندين:

p1: PNO < 300
p2: PNO >= 300

فإن كل الصفوف المدخلة والتي يضعها التوكيد في العلاقة NEW الجزئية من ASG تقوم بالمقارنتين:

$PROJ_1$ if PNO < 300
$PROJ_2$ if PNO >= 300

وهكذا باتصال واحد يتم التأكد من كل التقسيمات عند كل صف جديد يدخل إلى ASG.

أما إذا كان PROJ مقسما أفقيا وفق المسندين predicates:

p1: PNAME = 'CAD/CAM'
p2: PNAME ≠'CAD/CAM'

فإن كل صف في NEW من ASG سيتم مقارنتها مع كل التقسيمات $PROJ_i$، بخلاف الحالة السابقة.

فرض شروط السلامة

Enforcement of Integrity Constraints

يتم فرض شروط أو قيود السلامة حسب نوع التوكيد المستخدم:

في التوكيدات الفردية Individual assertions:

- يتم تخزين التوكيدات في المواقع مع الجداول.
- يتم إنشاء العلاقة +R و/أو العلاقة -R في كل موقع على حده.
- كل موقع يوافق أو يرفض التعديل (modification) اعتمادا على التوكيدات المناسبة له.

في التوكيدات الموجهة للمجموعة Set-oriented assertions: لدينا حالتين:

– شروط العلاقة الوحيدة Single-relation constraints:
  - يتم إنشاء العلاقة +R و/أو العلاقة -R ومن ثم إرسالها إلى كل موقع على حده.
  - يتم التحقق من كل موقع على حده.
  - إذا تم رفض موقع واحد، ربما يتم رفض بعض أو كل المواقع، حسب التحقق.

- شروط العلاقات المتعددة Multi-relation constraints (أكثر من علاقة):
  - يتم استرجاع جميع التوكيدات المترجمة (R, T, Ci).
  - لكل واحدة من التوكيدات Ci، تسترجع كل السجلات لكل موقع يمكن ان لا يحقق التوكيد.
  - عندما لا يحقق موقع معين شرط التوكيد، يتم رفضه.
  - ترسل الصفوف إلى كل موقع ثم تعود إلى المدير المركزي.

التوكيدات الإجمالية Aggregate assertions:

تكون معقدة في الغالب، وتعامل بطريقة أكثر قربا لفرض شروط السلامة بالنسبة للقيود ذات العلاقات المتعددة multi-relation constraints .